aws

ユーザーガイド

Amazon CloudWatch

Amazon CloudWatch: ユーザーガイド

Table of Contents

Amazon CloudWatch とは

Amazon CloudWatch は、Amazon Web Services (AWS) リソースと、AWS で実行されているアプリケーションをリアルタイムでモニターリングします。CloudWatch を使用してメトリクスを収集し、追跡できます。メトリクスとは、リソースやアプリケーションに関して測定できる変数です。

CloudWatch のホームページには、使用している AWS の各サービスに関するメトリクスが自動的に表示されます。さらに、カスタムダッシュボードを作成してカスタムアプリケーションのメトリクスを表示したり、選択したメトリクスのカスタムコレクションを表示したりできます。

メトリクスを監視し、しきい値を超過したときに通知を送信したり、モニターリングしているリソースを自動的に変更したりするアラームを作成できます。例えば、Amazon EC2 インスタンスの CPU 使用率およびディスク読み取り/書き込みをモニターリングし、そのデータを基に、増加する負荷を処理する追加のインスタンスを起動すべきかどうかを判断します。また、このデータを使用して、十分利用されていないインスタンスを停止することで、費用を節約することができます。

CloudWatch により、システム全体のリソース使用率、アプリケーションパフォーマンス、およびオペレーションの状態において可視性を得ることができます。

CloudWatch へのアクセス

次のいずれかの方法で CloudWatch にアクセスできます。

- Amazon CloudWatch コンソール – https://console.aws.amazon.com/cloudwatch/
- AWS CLI – 詳細については、「AWS Command Line Interface ユーザーガイド」の「AWS Command Line Interface のセットアップ」を参照してください。
- CloudWatch API – 詳細については、「Amazon CloudWatch API リファレンス」を参照してください。
- AWS SDK – 詳細については、Amazon Web Services 用ツールを参照してください。

関連 AWS サービス

Amazon CloudWatch と一緒に使用されるサービスは以下のとおりです。

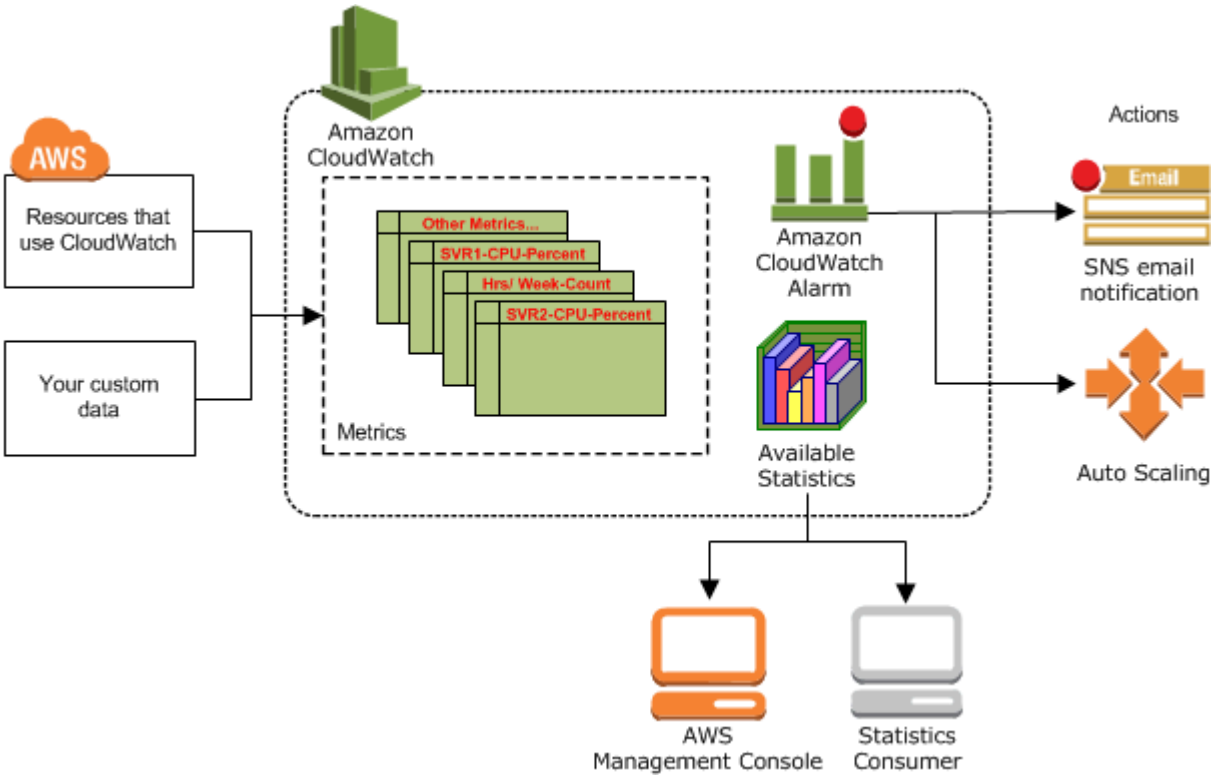
- Amazon Simple Notification Service (Amazon SNS) は、サブスクライブしているエンドポイントやクライアントへのメッセージ配信や送信を調整および管理します。アラームしきい値に達したと

きにメッセージを送信するには、CloudWatch で Amazon SNS を使用します。詳細については、「Amazon SNS 通知の設定」を参照してください。

- Amazon EC2 Auto Scaling により、ユーザー定義のポリシー、ヘルスステータスチェック、およびスケジュールに基づいて Amazon EC2 インスタンスを自動的に起動または終了します。Amazon EC2 Auto Scaling で CloudWatch アラームを使用して、EC2 インスタンスをオンデマンドでスケーリングできます。詳細については、「Amazon EC2 Auto Scaling ユーザーガイド」の「動的スケーリング」を参照してください。
- AWS CloudTrail は、アカウントの Amazon CloudWatch API 宛てのコール (AWS Management Console、AWS CLI などのサービスによって行われるコールを含む) のモニターリングを可能にします。CloudTrail ログ記録をオンにすると、CloudWatch は CloudTrail を設定したときに指定した Amazon S3 バケットにログファイルを書き込みます。詳細については、「AWS CloudTrail を使用した Amazon CloudWatch API コールのログ記録」を参照してください。
- AWS Identity and Access Management (IAM) は、AWS リソースへのユーザーアクセスを安全に管理するウェブサービスです。IAM により、どのユーザーがお客様の AWS リソースを使用できるか (認証)、それらのユーザーがどのリソースをどのような方法で使用できるか (承認) を制御できます。詳細については、「Amazon CloudWatch 用 Identity and Access Management」を参照してください。

Amazon CloudWatch の仕組み

Amazon CloudWatch は基本的にメトリクスリポジトリです。AWS のサービス (Amazon EC2 など) は、メトリクスをレポジトリに置き、これらのメトリクスを基に統計が取得されます。独自のカスタムメトリクスをレポジトリーに置いた場合も、それらのメトリクスを基に統計を取得できます。



メトリクスを使用して統計を計算し、そのデータをグラフ化して CloudWatch コンソールに表示できます。メトリクスを生成して CloudWatch に送信するその他の AWS リソースの詳細については、CloudWatch メトリクスを発行する AWS のサービス を参照してください。

一定の基準が満たされたときに Amazon EC2 インスタンスを停止、開始、または終了するアラームアクションを設定できます。また、ご自身の代わりに Amazon EC2 Auto Scaling および Amazon Simple Notification Service (Amazon SNS) アクションを開始するアラームを作成できます。CloudWatch アラームの作成の詳細については、「アラーム」を参照してください。

AWS クラウドコンピューティングリソースは、高可用性データセンター設備に収容されています。拡張性と信頼性を高めるため、各データセンターは、リージョンとして知られる特定の地域に配置されています。各リージョンは、障害が発生しても切り分けてできる限り安定性を高めるため、他のリージョンから完全に分離される設計になっています。メトリクスはリージョンに個別に保存されますが、CloudWatch クロスリージョン機能を使用して、異なるリージョンから統計を集計できます。詳細については、「クロスアカウントクロスリージョン CloudWatch コンソール」、および「Amazon Web Services 全般のリファレンス」のリージョンとエンドポイントについてのセクションを参照してください。

Amazon CloudWatch の概念

Amazon CloudWatch を理解し使用するために重要な用語と概念を、以下に示します。

- 名前空間
- メトリクス
- ディメンション
- 解像度
- 統計
- パーセンタイル
- アラーム

CloudWatch メトリクス、アラーム、API リクエスト、および E メール通知に関するサービスクォータについては、「CloudWatch service quotas」(CloudWatch サービスクォータ) を参照してください。

名前空間

名前空間は、CloudWatch メトリクスのコンテナです。異なる名前空間のメトリクスは相互に切り離されて、異なるアプリケーションのメトリクスが誤って同じ統計に集約されないようになっています。

デフォルトの名前空間はありません。CloudWatch に発行する各データポイントには、名前空間を指定する必要があります。メトリクスの作成時に名前空間名を指定できます。これらの名前には有効な ASCII 文字を含める必要があり、長さを 255 文字以下にする必要があります。使用可能な文字は、英数字 (0-9A-Za-z)、ピリオド (.)、ハイフン (-)、アンダースコア (_)、スラッシュ (/)、ハッシュ (#)、コロン (:)、スペース文字です。名前空間には、空白以外の文字を 1 つ以上含める必要があります。

AWS 名前空間では通常、命名規則 `AWS/service` が使用されます。例えば、Amazon EC2 は `AWS/EC2` 名前空間を使用します。AWS 名前空間のリストは、CloudWatch メトリクスを発行する AWS のサービス を参照してください。

メトリクス

メトリクスは CloudWatch での基本的な概念です。メトリクスは、CloudWatch に発行された時系列のデータポイントのセットを表します。メトリクスはモニターリング対象の変数と考え、データポ

イントは時間の経過と共に変数の値を表します。例えば、特定の EC2 インスタンスの CPU 使用率は、Amazon EC2 により提供される 1 つのメトリクスです。データポイント自体は、データの収集元のアプリケーションまたはビジネスアクティビティから生成されます。

デフォルトでは、多くの AWS のサービスはリソース (Amazon EC2 インスタンス、Amazon EBS ボリューム、Amazon RDS DB インスタンスなど) に対してメトリクスを無料で提供しています。料金については、Amazon EC2 インスタンスなど一部のリソースの詳細モニターリングを有効にしたり、独自のアプリケーションメトリクスを発行したりもできます。カスタムメトリクスには、データポイントを任意の順序や比率で追加できます。それらのデータポイントについての統計を、順序付けられた時系列データのセットとして取得できます。

メトリクスは作成されたリージョンにのみ存在します。メトリクスは削除できませんが、それらに対して新しいデータが発行されない場合、15 か月後に自動的に有効期限切れになります。15 か月以上経過したデータポイントは、ローリング方式で期限切れになります。つまり、新しいデータポイントが入ってくるたびに、古い 15 か月以上経過したものが削除されます。

メトリクスは名前、名前空間、0 以上のディメンションで一意に定義されます。メトリクスの各データポイントには、タイムスタンプと、(オプションで) 測定単位があります。CloudWatch から任意のメトリクスの統計情報を取得できます。

詳細については、「利用可能なメトリクスを表示する」および「カスタムメトリクスをパブリッシュする」を参照してください。

タイムスタンプ

各メトリクスデータポイントに、タイムスタンプが関連付けられている必要があります。タイムスタンプは、最大 2 週間前から最大 2 時間後になります。ユーザーがタイムスタンプを指定しない場合は、データポイントを受信した時間を基に、CloudWatch がタイムスタンプを作成します。

タイムスタンプは、完全な日付、時間、分、秒を含む `dateTime` オブジェクトです (たとえば、2016-10-31T23:59:59Z)。詳細については、「dateTime」を参照してください。必須ではありませんが、協定世界時 (UTC) を使用することをお勧めします。CloudWatch から統計を取得すると、すべての時刻は UTC になります。

CloudWatch アラームは、現在の UTC 時刻に基づいてメトリクスをチェックします。現在の UTC 時刻以外のタイムスタンプで CloudWatch にカスタムメトリクスが送信されると、アラームが Insufficient Data 状態を表示するか、アラームに遅延が生じる可能性があります。

メトリクスの保持

CloudWatch では、次のようにメトリクスデータを保持します。

- 期間が 60 秒未満のデータポイントは、3 時間使用できます。これらのデータポイントは高解像度カスタムメトリクスです。
- 期間が 60 秒 (1 分) のデータポイントは、15 日間使用できます。
- 期間が 300 秒 (5 分) のデータポイントは、63 日間使用できます。
- 期間が 3600 秒 (1 時間) のデータポイントは、455 日 (15 か月) 間使用できます。

最初は短い期間で発行されるデータポイントは、長期的なストレージのため一緒に集計されます。たとえば、1 分の期間でデータを収集する場合、データは 1 分の解像度で 15 日にわたり利用可能になります。15 日を過ぎてもこのデータはまだ利用できますが、集計され、5 分の解像度のみで取得可能になります。63 日を過ぎるとこのデータはさらに集計され、1 時間の解像度のみで利用できます。

> **Note**
>
> 過去 2 週間に新しいデータポイントがないメトリクスは、コンソールに表示されません。また、これらはコンソールの [すべてのメトリクス] タブの検索ボックスにメトリクス名やディメンション名を入力しても表示されず、list-metrics コマンドの結果でも返されません。これらのメトリクスを取得する最善の方法は、AWS CLI の get-metric-data コマンドまたは get-metric-statistics コマンドを使用することです。

ディメンション

ディメンションは、メトリクスのアイデンティティの一部である名前と値のペアです。1 メトリクスあたり最大 30 ディメンションを割り当てることができます。

各メトリクスには、それを表す固有の特徴があります。ディメンションはその特徴のカテゴリーと考えることができます。ディメンションは、統計プランの構造を設計するのに役立ちます。ディメンションは、メトリクスの一意の識別子の一部であるため、メトリクスに一意の名前と値のペアを追加するときは常に、そのメトリクスの新しいバリエーションを作成していることになります。

CloudWatch にデータを送信する AWS のサービスは、各メトリクスにディメンションを指定します。CloudWatch が返す結果にフィルタを掛ける際にディメンションを使用できます。たとえば、メトリクスを検索する際に `InstanceId` ディメンションを指定することにより、特定の EC2 インスタンスの統計を取得できます。

Amazon EC2 など特定の AWS のサービスが生成するメトリクスでは、CloudWatch は複数のディメンションにまたがるデータを集約する場合があります。例えば、ディメンションを指定せずに `AWS/`

EC2 名前空間にあるメトリクスを検索すると、CloudWatch は、指定されたメトリクスの全データを集約して、リクエストされた統計を作成します。CloudWatch はディメンション間ではカスタムメトリクスを集約しません。

ディメンションの組み合わせ

CloudWatch は、メトリクスのメトリクス名が同じ場合でも、ディメンションの一意の各組み合わせを別個のメトリクスとして扱います。明示的に発行したディメンションの組み合わせを用いてのみ、統計を取得できます。統計を取得するとき、名前空間、メトリクス名、ディメンションパラメータに、メトリクス作成時に使用されたのと同じ値を指定します。CloudWatch が集約に使用する開始時間と終了時間も指定できます。

たとえば、次のプロパティを使用して、DataCenterMetric 名前空間内で ServerStats という名前の 4 つの異なるメトリクスを発行するとします。

```
Dimensions: Server=Prod, Domain=Frankfurt, Unit: Count, Timestamp:
 2016-10-31T12:30:00Z, Value: 105
Dimensions: Server=Beta, Domain=Frankfurt, Unit: Count, Timestamp:
 2016-10-31T12:31:00Z, Value: 115
Dimensions: Server=Prod, Domain=Rio,       Unit: Count, Timestamp:
 2016-10-31T12:32:00Z, Value: 95
Dimensions: Server=Beta, Domain=Rio,       Unit: Count, Timestamp:
 2016-10-31T12:33:00Z, Value: 97
```

これらの 4 つのメトリクスのみを発行した場合、次のディメンションの組み合わせで統計を取得できます。

- `Server=Prod,Domain=Frankfurt`
- `Server=Prod,Domain=Rio`
- `Server=Beta,Domain=Frankfurt`
- `Server=Beta,Domain=Rio`

次のディメンションの統計を取得したり、ディメンションを指定しない場合に統計を取得したりすることはできません。(例外は、複数のメトリクスの統計を取得できる Metric Math の SEARCH 関数を使用することです。詳細については、「グラフで検索式を使用する」を参照してください。)

- `Server=Prod`
- `Server=Beta`

- `Domain=Frankfurt`
- `Domain=Rio`

解像度

各メトリクスは次のいずれかです。

- 詳細度が 1 分のデータを含む、標準の解像度
- 詳細度が 1 秒のデータを含む高解像度

AWS のサービスによって生成されたメトリクスは、デフォルトで標準解像度になります。カスタムメトリクスを発行するときは、標準解像度または高解像度のいずれかとして定義できます。高分解能のメトリクスをパブリッシュすると、CloudWatch はそれを 1 秒の分解能で保存します。ユーザーは、1 秒、5 秒、10 秒、30 秒、または 60 秒の倍数の期間でメトリクスを読み取り、取得できます。

高解像度メトリクスを使用すれば、アプリケーションの 1 分未満のアクティビティをより迅速に把握できます。`PutMetricData` がカスタムメトリクスを呼び出す場合、課金されることに注意してください。そのため、高解像度で `PutMetricData` を頻繁に呼び出すと、高額な料金が発生する可能性があります。CloudWatch の料金の詳細については、Amazon CloudWatch の料金をご覧ください。

高解像度メトリクスでアラームを設定する場合、10 秒または 30 秒の期間で高解像度アラームを指定するか、60 秒の倍数の期間で通常のアラームを設定できます。10 秒または 30 秒の期間の高解像度アラームでは、料金が高くなります。

統計

統計は、指定した期間のメトリクスデータの集計です。CloudWatch は、カスタムデータまたは他の AWS のサービスから CloudWatch に与えられたメトリクスのデータポイントを基に、統計を提供します。集約は、指定した期間内に、名前空間、メトリクス名、ディメンション、データポイントの測定単位を用いて行われます。

CloudWatch でサポートされている統計の詳細定義については、CloudWatch 統計定義 を参照してください。

単位

各統計には、測定単位があります。単位の例は、Bytes、Seconds、Count、Percent などです。CloudWatch がサポートするユニットの詳細なリストについては、「Amazon CloudWatch API リファレンス」の「MetricDatum」データ型を参照してください。

カスタムメトリクスを作成するときに単位を指定できます。単位を指定しない場合、CloudWatch は単位として None を使用します。単位は、データの概念的意味を与えるのに役立ちます。CloudWatch は内部的に単位に意味を持たせていませんが、他のアプリケーションでは、単位を基に意味のある情報を引き出すことができます。

測定単位を指定するメトリクスデータポイントは個別に集約されます。単位を指定せずに統計を取得する場合、CloudWatch が同じ単位のデータポイントをすべて 1 つに集約します。単位だけが異なっている 2 つのメトリクスがある場合、単位ごとに別々の 2 つのデータストリームが返されます。

期間

期間とは、特定の Amazon CloudWatch 統計に関連付けられた時間長です。各統計は、指定された期間に収集されたメトリクスデータの集約を表しています。期間は秒数で定義され、期間の有効な値は、1、5、10、30、または 60 の倍数になります。たとえば、6 分の期間を指定するには、期間の値として 360 を使用します。期間の長さを変えることで、データの集約方法を調整できます。デフォルト値は 60 秒間です。長さは最短で 1 秒にすることができ、デフォルト値の 60 秒を超える場合は、60 の倍数にする必要があります。

1 秒のストレージ解像度で定義したカスタムメトリクスのみが、1 分未満の解像度をサポートします。60 秒未満の期間を設定するオプションは常にコンソールで利用できますが、メトリクスの保存方法に合った期間を選択する必要があります。1 分未満の期間をサポートするメトリクスの詳細については、「高解像度のメトリクス」を参照してください。

統計を取得するとき、期間、開始時刻、終了時刻を指定できます。これらのパラメータでは、統計に関連する全体の時間長を決定します。開始時刻と終了時刻のデフォルト値では、過去 1 時間分の統計が得られます。開始時刻と終了時刻に指定した値により、CloudWatch により返される期間が決まります。たとえば、期間、開始時刻、終了時刻のデフォルト値を使用して統計を取得すると、過去 1 時間で毎分の集約された統計一式が返されます。10 分区切りで集約された統計を取得する場合は、期間を 600 に指定します。1 時間分の集約された統計の場合は、期間を 3600 に設定します。

一定期間にわたって統計が集計されると、その期間の開始時点に対応する時刻のタイムスタンプが付きます。たとえば、午後 7:00 ~ 午後 8:00 に集計されたデータには、午後 7:00 のタイムスタンプが付きます。さらに、午後 7:00 ~ 午後 8:00 に集計されたデータは午後 7:00 から表示され、この期間

中に CloudWatch がさらに多くのサンプルを集計するにつれて、集計データの値が変わる場合があります。

CloudWatch アラームでは、期間も重要です。特定のメトリクスをモニターリングするアラームを作成したなら、そのメトリクスと指定したしきい値を比較するよう CloudWatch に依頼していることになります。ユーザーは、CloudWatch がその比較を行う方法を広範囲に制御できます。どれくらいの期間で比較するかだけでなく、結論までに使用する評価期間数も設定できます。例えば、3 つの評価期間を設定すると、CloudWatch は 3 つのデータポイントのウィンドウを比較します。最も古い期間が超過し、その他の期間が超過または不足している場合にのみ、CloudWatch は通知を送ります。

集計

統計の取得時に指定した期間の長さに従って、Amazon CloudWatch は統計を集約します。同一または類似したタイムスタンプを使って、望む数のデータポイントをパブリッシュできます。CloudWatch は、指定された期間の長さに従って集約します。CloudWatch はリージョン間でデータを自動的に集約しませんが、メトリクスの計算を使用して、異なるリージョンのメトリクスを集約できます。

同じタイムスタンプだけでなく、同じ名前空間とディメンションを共有するメトリクスのデータポイントをパブリッシュできます。CloudWatch は、それらのデータポイントの集約された統計を返します。また、どのタイムスタンプでも、同じメトリクスまたは異なるメトリクスの複数のデータポイントをパブリッシュすることもできます。

大きいデータセットの場合、統計セットという事前集約されたデータセットを挿入できます。統計セットにより、CloudWatch に多数のデータポイントの Min (最小値)、Max (最大値)、Sum (合計)、SampleCount (サンプルカウント) を与えます。これは、1 分間に何回もデータを収集する必要がある場合によく使用されます。たとえば、ウェブページのリクエストレイテンシーに関するメトリクスがあるとします。ウェブページがヒットされるたびにデータを発行することは意味を成しません。そこで、そのウェブページの全ヒットのレイテンシーを収集してそのデータを毎分集約し、その統計セットを CloudWatch に送信することができます。

Amazon CloudWatch は、メトリクスのソースを区別しません。ソースが異なっていても、名前空間とディメンションが同じメトリクスがパブリッシュされれば、CloudWatch はこれを 1 つのメトリクスとして扱います。これは、分散型の、拡大縮小されたシステムのサービスメトリクスで有用です。たとえば、ウェブサーバーのアプリケーションのすべてのホストが処理するリクエストのレイテンシーを表すメトリクスを 1 つにまとめてパブリッシュできます。CloudWatch はこれらを 1 つのメトリクスとして扱い、アプリケーション全体ですべてのリクエストの最小値、最大値、平均、合計の統計を取得できるようにします。

パーセンタイル

パーセンタイルは、データセットにおける値の相対的な位置を示します。たとえば、95 パーセンタイルは、95 パーセントのデータがこの値を下回っており、5 パーセントのデータがこの値を上回っていることを意味します。パーセンタイルにより、メトリクスデータの分布をよく理解することができます。

パーセンタイルは、異常を分離するためによく使用されます。正規分布では、95 パーセントのデータが平均から 2 標準偏差以内に収まっており、99.7 パーセントのデータが平均から 3 標準偏差以内に収まっています。3 標準偏差に収まらないデータは、平均値とかけ離れているため、多くの場合異常と見なされます。たとえば、お客様の満足度を高めるため、EC2 インスタンスの CPU 使用率をモニターリングしているとします。平均をモニターリングした場合、異常が見えなくなる可能性があります。最大をモニターリングした場合、1 つの異常のために正しい結果が見えなくなる可能性があります。パーセンタイルを使用すると、CPU 使用率の 95 パーセンタイルをモニターリングし、負荷が異常に高いインスタンスをチェックできます。

CloudWatch メトリクスの中には、統計としてパーセンタイルがサポートされています。これらのメトリクスの場合、他の CloudWatch 統計 (平均、最小、最大、合計) を使用するのと同じように、パーセンタイルを使用してシステムとアプリケーションをモニターリングできます。たとえば、アラームを作成するとき、パーセンタイルを統計関数として使用できます。小数点以下最大 10 桁のパーセンタイルを指定できます (p95.0123456789 など)。

カスタムメトリクスの要約されていない raw データポイントを発行するかぎり、カスタムメトリクスでパーセンタイル統計を利用できます。パーセンタイル統計は、メトリクス値が負の数値のメトリクスに対して使用することはできません。

CloudWatch は、raw データポイントを使用してパーセンタイルを計算します。代わりに統計セットを使用してデータを発行する場合は、以下の条件のいずれか真である場合のみ、このデータのパーセンタイル統計を取得できます。

- 統計セットの SampleCount 値は 1 で、Min、Max、Sum はすべて等しくなります。
- Min と Max は等しく、Sum は Min に SampleCount を乗算した値に等しくなります。

次の AWS サービスには、パーセンタイル統計をサポートするメトリクスが含まれています。

- API Gateway
- Application Load Balancer
- Amazon EC2

- Elastic Load Balancing
- Kinesis
- Amazon RDS

CloudWatch では、トリミングされた平均値やその他のパフォーマンス統計もサポートしており、パーセンタイルと同様の使用が可能です。詳細については、「CloudWatch 統計定義」を参照してください。

アラーム

アラームを使用すると、アクションを自動的に開始することができます。アラームは、指定した期間の単一のメトリクスを監視し、一定期間におけるしきい値とメトリクスの値の関係性に基づいて、1 つ以上の指定されたアクションを実行します。アクションは、Amazon SNS のトピックまたは Auto Scaling のポリシーに送信される通知です。アラームはダッシュボードに追加することもできます。

アラームは、持続している状態変化に対してのみアクションを呼び出します。CloudWatch アラームは、特定の状態にあるという理由だけではアクションを呼び出しません。状態が変わって、変わった状態が指定期間にわたって維持される必要があります。

アラームを作成するときは、メトリクスの解像度以上のアラームのモニターリング期間を選択します。例えば、Amazon EC2 の基本モニターリングでは、5 分ごとにインスタンスのメトリクスが提供されます。基本モニターリングのメトリクスにアラームを設定する場合、少なくとも 300 秒 (5 分) の期間を選択します。Amazon EC2 の詳細モニターリングでは、1 分の解像度でインスタンスのメトリクスが提供されます。詳細モニターリングのメトリクスにアラームを設定する場合、少なくとも 60 秒 (1 分) の期間を選択します。

高解像度メトリクスでアラームを設定する場合、10 秒または 30 秒の期間で高解像度アラームを指定するか、60 秒の倍数の期間で通常のアラームを設定できます。高解像度のアラームには高い料金が発生します。高解像度メトリクスの詳細については、「カスタムメトリクスをパブリッシュする」を参照してください。

詳細については、Amazon CloudWatch でのアラームの使用およびグラフのメトリクスからアラームを作成するを参照してください。

請求とコスト

CloudWatch の料金の完全な詳細については、「Amazon CloudWatch の料金」を参照してください。

請求書を分析し、コストを最適化および削減するのに役立つ情報については、「CloudWatch の請求とコスト」を参照してください。

Amazon CloudWatch のリソース

このサービスを利用する際に役立つ関連リソースは以下の通りです。

リソース	説明
Amazon CloudWatch のよくある質問	本製品に関して開発者からよく寄せられる上位の質問です。
AWS デベロッパーセンター	資料、コード例、リリースノートをはじめとする、AWS ベースの革新的なアプリケーション開発に役立つさまざまな情報が収められた、中心的起点となるリソースセンターです。
AWS Management Console	コンソールでは、プログラミングを行うことなく、Amazon CloudWatch とその他の AWS のサービスのほとんどの機能を実行できます。
Amazon CloudWatch ディスカッションフォーラム	Amazon CloudWatch に関する技術的な質疑応答の場であるデベロッパー向けのコミュニティフォーラムです。
AWS Support	AWS Support のケースを作成して管理するためのハブです。フォーラム、技術上のよくある質問、サービスヘルスステータス、AWS Trusted Advisor などの便利なリソースへのリンクも含まれています。
Amazon CloudWatch 製品情報	Amazon CloudWatch に関する情報のメインのウェブページです。
お問い合わせ	AWS の請求、アカウント設定その他に関する連絡先です。

準備作業

Amazon CloudWatch を使用するには、AWS アカウントが必要です。AWS アカウントがあれば、Amazon EC2 などのサービスを利用して、CloudWatch コンソール (ポイントしてクリックするウェブベースのインターフェイス) で表示できるメトリクスを生成できます。加えて、AWS コマンドラインインターフェイス (CLI) をインストールし、設定することができます。

AWS アカウントへのサインアップ

AWS アカウントがない場合は、以下のステップを実行して作成します。

AWS アカウントにサインアップするには

1. https://portal.aws.amazon.com/billing/signup を開きます。
2. オンラインの手順に従います。

   サインアップ手順の一環として、通話呼び出しを受け取り、電話キーパッドで検証コードを入力するように求められます。

   AWS アカウントにサインアップすると、AWS アカウントのルートユーザーが作成されます。ルートユーザーには、アカウントのすべてのAWS のサービスとリソースへのアクセス権があります。セキュリティのベストプラクティスとして、ユーザーに管理アクセスを割り当て、ルートユーザーのみを使用してルートユーザーアクセスが必要なタスクを実行してください。

サインアップ処理が完了すると、AWS からユーザーに確認メールが送信されます。https://aws.amazon.com/ の [マイアカウント] を選んで、いつでもアカウントの現在のアクティビティを表示し、アカウントを管理できます。

管理アクセスを持つユーザーを作成する

AWS アカウント にサインアップしたら、AWS アカウントのルートユーザー をセキュリティで保護し、AWS IAM Identity Center を有効にして、管理ユーザーを作成します。これにより、日常的なタスクにルートユーザーを使用しないようにします。

AWS アカウントのルートユーザーをセキュリティで保護する

1. [ルートユーザー] を選択し、AWS アカウントのメールアドレスを入力して、アカウント所有者として AWS Management Console にサインインします。次のページでパスワードを入力します。

   ルートユーザーを使用してサインインする方法については、AWS サインイン ユーザーガイドの「ルートユーザーとしてサインインする」を参照してください。
2. ルートユーザーの多要素認証 (MFA) を有効にします。

   手順については、IAM ユーザーガイドの「AWS アカウント のルートユーザーの仮想 MFA デバイスを有効にする (コンソール)」を参照してください。

管理アクセスを持つユーザーを作成する

1. IAM アイデンティティセンターを有効にします。

   手順については、「AWS IAM Identity Center ユーザーガイド」の「AWS IAM Identity Center の有効化」を参照してください。
2. IAM アイデンティティセンターで、ユーザーに管理アクセスを付与します。

   IAM アイデンティティセンターディレクトリ をアイデンティティソースとして使用するチュートリアルについては、「AWS IAM Identity Center ユーザーガイド」の「デフォルト IAM アイデンティティセンターディレクトリを使用したユーザーアクセスの設定」を参照してください。

管理アクセス権を持つユーザーとしてサインインする

- IAM アイデンティティセンターのユーザーとしてサインインするには、IAM アイデンティティセンターのユーザーの作成時に E メールアドレスに送信されたサインイン URL を使用します。

  IAM Identity Center ユーザーを使用してサインインする方法については、AWS サインイン ユーザーガイドの「AWS アクセスポータルにサインインする」を参照してください。

追加のユーザーにアクセス権を割り当てる

1. IAM アイデンティティセンターで、最小特権のアクセス許可を適用するというベストプラクティスに従ったアクセス許可セットを作成します。

手順については、「AWS IAM Identity Center ユーザーガイド」の「権限設定を作成する」を参照してください。

2. グループにユーザーを割り当て、そのグループにシングルサインオンアクセス権を割り当てます。

   手順については、「AWS IAM Identity Center ユーザーガイド」の「グループの参加」を参照してください。

Amazon CloudWatch コンソールにサインインする

Amazon CloudWatch コンソールにサインインするには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. 必要に応じて、ナビゲーションバーを使い、リージョンを AWS リソースがあるリージョンに変更します。
3. CloudWatch コンソールを初めて使用する場合でも、[Your Metrics] (自分のメトリクス) にメトリクスが表示されることがあります。これは、Amazon CloudWatch に無料で自動的にメトリクスをプッシュする AWS 製品を使用したことがある場合です。その他のサービスでは、メトリクスを有効にする必要があります。

   作成済みのアラームがない場合、[Your Alarms] セクションに [Create Alarm] ボタンが表示されます。

AWS CLI をセットアップする

AWS CLI または Amazon CloudWatch CLI を使用して、CloudWatch コマンドを実行できます。AWS CLI は CloudWatch CLI を置き換えることに注意してください。AWS CLI にのみ新しい CloudWatch 機能が含まれています。

AWS CLI をインストールして設定する方法については、AWS Command Line Interface ユーザーガイドの「AWS Command Line Interface のセットアップ」を参照してください。

Amazon CloudWatch CLI をインストールおよび設定する方法については、Amazon CloudWatch CLI リファレンスの「コマンドラインインターフェイスをセットアップする」を参照してください。

Amazon CloudWatch の開始方法

CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。

CloudWatch 概要のホームページが表示されます。

概要には以下の項目が表示されます。各項目は自動的に更新されます。

- AWS サービス別のアラームには、アカウントで使用している AWS サービスのリストと、それらのサービスのアラームの状態が表示されます。その横に、アカウントのアラームが 2 つまたは 4 つ表示されます。この数は、使用する AWS サービスの数によって異なります。アラームとして ALARM 状態、または最近変わった状態が表示されます。

  上部のこれらの領域により、各サービスのアラーム状態と最近変化した状態を確認し、AWS のサービスの正常性を迅速に評価するのに役立ちます。これは問題のモニタリングと迅速な診断に役立ちます。
- これらの領域の下には、デフォルトのダッシュボード (存在する場合) があります。デフォルトのダッシュボードは、CloudWatch-Default という名前のカスタムダッシュボードです。カスタムダッシュボードでは、簡単に独自のサービスやアプリケーションに関するメトリクスを概要ページに追加したり、AWS のサービスで詳細にモニタリングする主要メトリクスを増やしたりできます。

> **Note**
>
> CloudWatch ホームページの自動ダッシュボードには、アカウントが CloudWatch クロスアカウントオブザーバビリティ用に設定されたモニタリングアカウントであっても、現在のアカウントの情報のみが表示されます。カスタムのクロスアカウントダッシュボードの作成についての詳細は、「CloudWatch クロスアカウントオブザーバビリティダッシュボード」を参照してください。

この概要から、複数の AWS サービスからのメトリクスのクロスサービスダッシュボードを表示したり、特定のリソースグループまたは特定の AWS サービスに絞って表示したりできます。これにより、関心があるリソースのサブセットを絞り込んで確認できます。詳細については、次のセクションを参照してください。

1 つのサービスの自動構築済みダッシュボードを表示する

1 つのサービスの自動構築済みダッシュボードを表示するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。

   ホームページに表示されます。
2. 左のナビゲーションペインの [ダッシュボード] を選択します。
3. [自動ダッシュボード]タブを選択し、表示するサービスを選択します。
4. このサービスのアラームの表示に切り替えるには、サービス名が現在表示されている画面の上部近くにある [アラーム状態]、[データ不足]、または [OK] のチェックボックスをオンにします。
5. メトリクスを表示する際に、いくつかの方法で特定のメトリクスに絞ることができます。

   a. いずれかのグラフのメトリクスに関する詳細を表示するには、グラフ上にマウスを移動し、アクションアイコンを選択して [メトリクスを表示] を選択します。

      グラフが新しいタブで開き、関連するメトリクスがグラフの下に一覧表示されます。このグラフの表示はカスタマイズできます。表示されるメトリクスやリソースを変更し、統計や期間などの要因を変更することで、現在の状況を詳しく理解できます。

   b. グラフに表示される時間範囲からログイベントを表示できます。これにより、メトリクスの予期しない変更を発生させたインフラストラクチャ内のイベントを検出できます。

      ログイベントを表示するには、グラフ上にマウスを移動し、アクションアイコンを選択して [View in logs (ログを表示)] を選択します。

CloudWatch Logs ビューが新しいタブで開き、ロググループが一覧表示されます。元のグラフに表示された時間範囲内に発生したログイベントのいずれかのログイベントを表示するには、そのロググループを選択します。

6. アラームを表示する際に、いくつかの方法で特定のアラームに絞ることができます。

   - アラームの詳細を表示するには、アラーム上にマウスを移動し、アクションアイコンを選択して [アラームで表示] を選択します。

     アラームビューが新しいタブで開き、アラームのリストと、選択したアラームに関する詳細が表示されます。このアラームの履歴を表示するには、[履歴] タブを選択します。

7. アラームは常に 1 分に 1 回の頻度で更新されます。表示を更新するには、画面の右上にある更新アイコン (2 つの湾曲した矢印) を選択します。画面でアラーム以外の項目の自動更新の頻度を変更するには、更新アイコンの横にある下矢印を選択し、更新の頻度を選択します。自動更新をオフにすることもできます。

8. 現在表示されているすべてのグラフとアラームの時間範囲を変更するには、画面の上部にある [Time range] (時間範囲) の横で目的の範囲を選択します。デフォルトで表示される時間範囲より多くのオプションから選択するには、[カスタム] を選択します。

9. クロスサービスダッシュボードに戻るには、現在絞り込んでいるサービスが表示されている画面の上部のリストで [概要] を選択します。

   または、任意のビューで、画面の上部にある [CloudWatch] を選択してすべてのフィルターをクリアし、概要ページに戻ります。

構築済みのクロスサービスダッシュボードを参照する

クロスサービスダッシュボード画面に切り替えて、使用しているすべての AWS のサービスのダッシュボードを操作できます。CloudWatch コンソールでは、ダッシュボードがアルファベット順に表示され、各ダッシュボードに 1 つまたは 2 つの主要なメトリクスが表示されます。

> **Note**
>
> 5 つ以上の AWS のサービスを利用している場合、CloudWatch コンソールは [Overview] (概要) 画面にクロスサービスダッシュボードを表示しません。

クロスサービスダッシュボードを開くには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。

   [Overview] (概要) 画面に移動します。

2. [Overview] (概要) 画面から、[Overview] (概要) と表示されているドロップダウンを選択し、[Cross service dashboard] (クロスサービスダッシュボード) を選択します。

   クロスサービスのダッシュボード画面に移動します。

3. (オプション) 元のインターフェイスを使用している場合は、[Cross-service dashboard] (クロスサービスダッシュボード) セクションまでスクロールし、[View Cross-service dashboard] (クロスサービスダッシュボードを表示) を選択します。

   クロスサービスのダッシュボード画面に移動します。

4. 特定のサービスに絞るには 2 つの方法があります。

   a. 表示するサービスの主要メトリクスの数を増やすには、クロスサービスダッシュボードが現在表示されている画面の上部にあるリストからサービス名を選択します。または、サービス名の横にある [View Service dashboard (サービスダッシュボードの表示)] を選択します。

      サービスの自動ダッシュボードに、このサービスの追加のメトリクスが表示されます。さらに、一部のサービスでは、サービスダッシュボードの下部にこのサービスと関連するリソースが表示されます。これらのリソースのいずれかをサービスのコンソールで選択し、このリソースをさらに詳しく調査できます。

   b. サービスに関連するすべてのアラームを確認するには、画面の右側でサービス名の横にあるボタンを選択します。このボタン上に、このサービスで作成したアラームの数と、ALARM 状態になっているアラームがあるかどうかが表示されます。

      アラームを表示する際に、設定 (ディメンション、しきい値、期間など) が似ている複数のアラームは単一のグラフに表示されます。

      次に、各アラームの詳細とアラーム履歴を確認できます。これを行うには、アラームのグラフ上にマウスを移動し、アクションアイコンを選択して [アラームで表示] を選択します。

      アラームビューが新しいブラウザタブで開き、アラームのリストと、選択したアラームに関する詳細が表示されます。このアラームの履歴を表示するには、[履歴] タブを選択します。

5. 特定のリソースグループのリソースに絞ることができます。これを行うには、ページの上部に表示されているすべてのリソースのリストからリソースグループを選択します。

詳細については、「リソースグループの事前構築済みダッシュボードを表示する」を参照してください。

6. 現在表示されているすべてのグラフとアラームの時間範囲を変更するには、画面の上部にある [時間範囲] の横で目的の範囲を選択します。デフォルトで表示される時間範囲より多くのオプションから選択するには、[カスタム] を選択します。
7. アラームは常に 1 分に 1 回の頻度で更新されます。表示を更新するには、画面の右上にある更新アイコン (2 つの湾曲した矢印) を選択します。画面でアラーム以外の項目の自動更新の頻度を変更するには、更新アイコンの横にある下矢印を選択し、目的の更新の頻度を選択します。自動更新をオフにすることもできます。

クロスサービスダッシュボードからサービスを削除する

クロスサービスダッシュボードからサービスのメトリクスを削除できます。これにより、詳細にモニタリングする必要があるサービスに絞ってクロスサービスダッシュボードに表示できます。

クロスサービスダッシュボードからサービスを削除しても、アラームビューには、このサービスのアラームが依然として表示されます。

クロスサービスダッシュボードからサービスのメトリクスを削除するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。

   ホームページに表示されます。
2. ページの上部の [概要] で、削除するサービスを選択します。

   ビューが変わり、そのサービスのメトリクスのみが表示されます。
3. [アクション] を選択し、[Show on cross service dashboard (クロスサービスダッシュボードに表示)] の横にあるチェックボックスをオフにします。

リソースグループの事前構築済みダッシュボードを表示する

単一のリソースグループのメトリクスとアラームに絞って表示できます。リソースグループでは、タグを使用してプロジェクトを整理したり、アーキテクチャーのサブセットに注目したり、本番稼働用環境と開発用環境を区別したりできます。また、CloudWatch の概要で、これらのリソースグループのいずれかに絞って調査できます。詳細については、「AWS Resource Groups とは」を参照してください。

リソースグループに絞ると、表示が変わり、このリソースグループの一部としてタグ付けしたリソースと関連するサービスのみが表示されます。最近のアラームの領域には、このリソースグループの一部であるリソースに関連付けられているアラームのみが表示されます。さらに、ダッシュボードを CloudWatch-Default-ResourceGroupName という名前で作成している場合、ダッシュボードは [デフォルトダッシュボード] 領域に表示されます。

AWS の単一のサービスと 1 つのリソースグループの両方に同時に絞ることで、詳しく調査できます。次の手順では、1 つのリソースグループに絞る方法のみを示します。

単一のリソースグループに絞るには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. すべてのリソースが表示されているページの上部で、リソースグループを選択します。
3. このリソースグループに関連するより多くのメトリクスを表示するには、画面の下部にある [View cross service dashboard (クロスサービスダッシュボードの表示)] を選択します。

   クロスサービスダッシュボードが開き、このリソースグループに関連するサービスのみが表示されます。サービスごとに 1 つまたは 2 つの主要メトリクスが表示されます。
4. 現在表示されているすべてのグラフとアラームの時間範囲を変更するには、画面の上部にある [時間範囲] で目的の範囲を選択します。デフォルトで表示される時間範囲より多くのオプションから選択するには、[カスタム] を選択します。
5. アラームは常に 1 分に 1 回の頻度で更新されます。表示を更新するには、画面の右上にある更新アイコン (2 つの湾曲した矢印) を選択します。画面でアラーム以外の項目の自動更新の頻度を変更するには、更新アイコンの横にある下矢印を選択し、更新の頻度を選択します。自動更新をオフにすることもできます。
6. アカウントのすべてのリソースに関する情報を再び表示するには、リソースグループの名前が現在表示されている画面の上部で、[すべてのリソース] を選択します。

構築済みのクロスサービスダッシュボードを参照する

クロスサービスダッシュボード画面に切り替えて、使用しているすべての AWS のサービスのダッシュボードを操作できます。CloudWatch コンソールでは、ダッシュボードがアルファベット順に表示され、各サービスの 1 つまたは 2 つの主要なメトリクスが表示されます。

> **Note**
>
> 5 つ以上の AWS のサービスを利用している場合、CloudWatch コンソールは [Overview] (概要) 画面にクロスサービスダッシュボードを表示しません。

クロスサービスダッシュボードを開くには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。

   [Overview] (概要) 画面に移動します。

2. [Overview] (概要) 画面から、[Overview] (概要) と表示されているドロップダウンを選択し、[Cross service dashboard] (クロスサービスダッシュボード) を選択します。

   クロスサービスのダッシュボード画面に移動します。

3. (オプション) 元のインターフェイスを使用している場合は、[Cross-service dashboard] (クロスサービスダッシュボード) セクションまでスクロールし、[View Cross-service dashboard] (クロスサービスダッシュボードを表示) を選択します。

   クロスサービスのダッシュボード画面に移動します。

4. 特定のサービスに絞るには 2 つの方法があります。

   a. 表示するサービスの主要メトリクスの数を増やすには、クロスサービスダッシュボードが現在表示されている画面の上部にあるリストからサービス名を選択します。または、サービス名の横にある [View Service dashboard (サービスダッシュボードの表示)] を選択します。

      サービスの自動ダッシュボードに、このサービスの追加のメトリクスが表示されます。さらに、一部のサービスでは、サービスダッシュボードの下部にこのサービスと関連するリソースが表示されます。これらのリソースのいずれかをサービスのコンソールで選択し、このリソースをさらに詳しく調査できます。

   b. サービスに関連するすべてのアラームを確認するには、画面の右側でサービス名の横にあるボタンを選択します。このボタン上に、このサービスで作成したアラームの数と、ALARM 状態になっているアラームがあるかどうかが表示されます。

      アラームを表示する際に、設定 (ディメンション、しきい値、期間など) が似ている複数のアラームは単一のグラフに表示されます。

次に、各アラームの詳細とアラーム履歴を確認できます。これを行うには、アラームのグラフ上にマウスを移動し、アクションアイコンを選択して [アラームで表示] を選択します。

アラームビューが新しいブラウザタブで開き、アラームのリストと、選択したアラームに関する詳細が表示されます。このアラームの履歴を表示するには、[履歴] タブを選択します。

5. 特定のリソースグループのリソースに絞ることができます。これを行うには、ページの上部に表示されているすべてのリソースのリストからリソースグループを選択します。

   詳細については、「リソースグループの事前構築済みダッシュボードを表示する」を参照してください。

6. 現在表示されているすべてのグラフとアラームの時間範囲を変更するには、画面の上部にある [時間範囲] の横で目的の範囲を選択します。デフォルトで表示される時間範囲より多くのオプションから選択するには、[カスタム] を選択します。

7. アラームは常に 1 分に 1 回の頻度で更新されます。表示を更新するには、画面の右上にある更新アイコン (2 つの湾曲した矢印) を選択します。画面でアラーム以外の項目の自動更新の頻度を変更するには、更新アイコンの横にある下矢印を選択し、目的の更新の頻度を選択します。自動更新をオフにすることもできます。

クロスサービスダッシュボードからサービスを削除する

クロスサービスダッシュボードからサービスのメトリクスを削除できます。これにより、詳細にモニタリングする必要があるサービスに絞ってクロスサービスダッシュボードに表示できます。

クロスサービスダッシュボードからサービスを削除しても、アラームビューには、このサービスのアラームが依然として表示されます。

クロスサービスダッシュボードからサービスのメトリクスを削除するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。

   ホームページに表示されます。

2. ページの上部の [概要] で、削除するサービスを選択します。

   ビューが変わり、そのサービスのメトリクスのみが表示されます。

3. [アクション] を選択し、[Show on cross service dashboard (クロスサービスダッシュボードに表示)] の横にあるチェックボックスをオフにします。

1 つの AWS サービスのビルド済みダッシュボードを表示する

CloudWatch ホームページでは、1 つの AWS サービスを表示して、詳細をご覧いただけます。AWS の単一のサービスと 1 つのリソースグループの両方に同時に絞ることで、詳しく調査できます。次の手順では、AWS のサービスに絞る方法のみを示します。

単一のサービスに絞るには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。

   ホームページに表示されます。
2. [概要] について、現在 [概要] がドロップダウンメニューに表示されている場合は、[サービスダッシュボード] を選択します。
3. 絞り込むサービスを選択します。

   表示が変わり、選択したサービスの主要メトリクスのグラフが表示されます。
4. このサービスのアラームの表示に切り替えるには、サービス名が現在表示されている画面の上部近くにある [アラーム状態]、[データ不足]、または [OK] のチェックボックスをオンにします。
5. メトリクスを表示する際に、いくつかの方法で特定のメトリクスに絞ることができます。

   a. いずれかのグラフのメトリクスに関する詳細を表示するには、グラフ上にマウスを移動し、アクションアイコンを選択して [メトリクスを表示] を選択します。

      グラフが新しいタブで開き、関連するメトリクスがグラフの下に一覧表示されます。このグラフの表示はカスタマイズできます。表示されるメトリクスやリソースを変更し、統計や期間などの要因を変更することで、現在の状況を詳しく理解できます。

   b. グラフに表示される時間範囲からログイベントを表示できます。これにより、メトリクスの予期しない変更を発生させたインフラストラクチャ内のイベントを検出できます。

      ログイベントを表示するには、グラフ上にマウスを移動し、アクションアイコンを選択して [View in logs (ログを表示)] を選択します。

      CloudWatch Logs ビューが新しいタブで開き、ロググループが一覧表示されます。元のグラフに表示された時間範囲内に発生したログイベントのいずれかのログイベントを表示するには、そのロググループを選択します。
6. アラームを表示する際に、いくつかの方法で特定のアラームに絞ることができます。

- アラームの詳細を表示するには、アラーム上にマウスを移動し、アクションアイコンを選択して [アラームで表示] を選択します。

  アラームビューが新しいタブで開き、アラームのリストと、選択したアラームに関する詳細が表示されます。このアラームの履歴を表示するには、[履歴] タブを選択します。

7. アラームは常に 1 分に 1 回の頻度で更新されます。表示を更新するには、画面の右上にある更新アイコン (2 つの湾曲した矢印) を選択します。画面でアラーム以外の項目の自動更新の頻度を変更するには、更新アイコンの横にある下矢印を選択し、更新の頻度を選択します。自動更新をオフにすることもできます。
8. 現在表示されているすべてのグラフとアラームの時間範囲を変更するには、画面の上部にある [Time range] (時間範囲) の横で目的の範囲を選択します。デフォルトで表示される時間範囲より多くのオプションから選択するには、[カスタム] を選択します。
9. クロスサービスダッシュボードに戻るには、現在絞り込んでいるサービスが表示されている画面の上部のリストで [概要] を選択します。

   または、任意のビューで、画面の上部にある [CloudWatch] を選択してすべてのフィルターをクリアし、概要ページに戻ります。

リソースグループの事前構築済みダッシュボードを表示する

単一のリソースグループのメトリクスとアラームに絞って表示できます。リソースグループでは、タグを使用してプロジェクトを整理したり、アーキテクチャーのサブセットに注目したり、本番稼働用環境と開発用環境を区別したりできます。また、CloudWatch の概要で、これらのリソースグループのいずれかに絞って調査できます。詳細については、「AWS Resource Groups とは」を参照してください。

リソースグループに絞ると、表示が変わり、このリソースグループの一部としてタグ付けしたリソースと関連するサービスのみが表示されます。最近のアラームの領域には、このリソースグループの一部であるリソースに関連付けられているアラームのみが表示されます。さらに、ダッシュボードを CloudWatch-Default-ResourceGroupName という名前で作成している場合、ダッシュボードは [デフォルトダッシュボード] 領域に表示されます。

AWS の単一のサービスと 1 つのリソースグループの両方に同時に絞ることで、詳しく調査できます。次の手順では、1 つのリソースグループに絞る方法のみを示します。

単一のリソースグループに絞るには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. すべてのリソースが表示されているページの上部で、リソースグループを選択します。
3. このリソースグループに関連するより多くのメトリクスを表示するには、画面の下部にある [View cross service dashboard (クロスサービスダッシュボードの表示)] を選択します。

   クロスサービスダッシュボードが開き、このリソースグループに関連するサービスのみが表示されます。サービスごとに 1 つまたは 2 つの主要メトリクスが表示されます。
4. 現在表示されているすべてのグラフとアラームの時間範囲を変更するには、画面の上部にある [時間範囲] で目的の範囲を選択します。デフォルトで表示される時間範囲より多くのオプションから選択するには、[カスタム] を選択します。
5. アラームは常に 1 分に 1 回の頻度で更新されます。表示を更新するには、画面の右上にある更新アイコン (2 つの湾曲した矢印) を選択します。画面でアラーム以外の項目の自動更新の頻度を変更するには、更新アイコンの横にある下矢印を選択し、更新の頻度を選択します。自動更新をオフにすることもできます。
6. アカウントのすべてのリソースに関する情報を再び表示するには、リソースグループの名前が現在表示されている画面の上部で、[すべてのリソース] を選択します。

CloudWatch の請求とコスト

このセクションでは、Amazon CloudWatch の機能でどのようにコストが発生するかについて説明します。また、CloudWatch のコストを分析、最適化、削減するのに役立つ方法についても説明します。このセクション全体を通して、CloudWatch 機能を説明する際に料金について言及することがあります。料金の詳細については、「Amazon CloudWatch の料金」を参照してください。

トピック

- Cost Explorer で CloudWatch のコストと使用状況データを分析する
- CloudWatch のコストと使用状況データを AWS Cost and Usage Report と Athena で分析する
- コスト削減と最適化のためのベストプラクティス

Cost Explorer で CloudWatch のコストと使用状況データを分析する

AWS Cost Explorer では、CloudWatch を含む AWS のサービス のコストと使用状況のデータを時間の経過を追って可視化して分析できます。詳細については、「AWS Cost Explorer の開始方法」を参照してください。

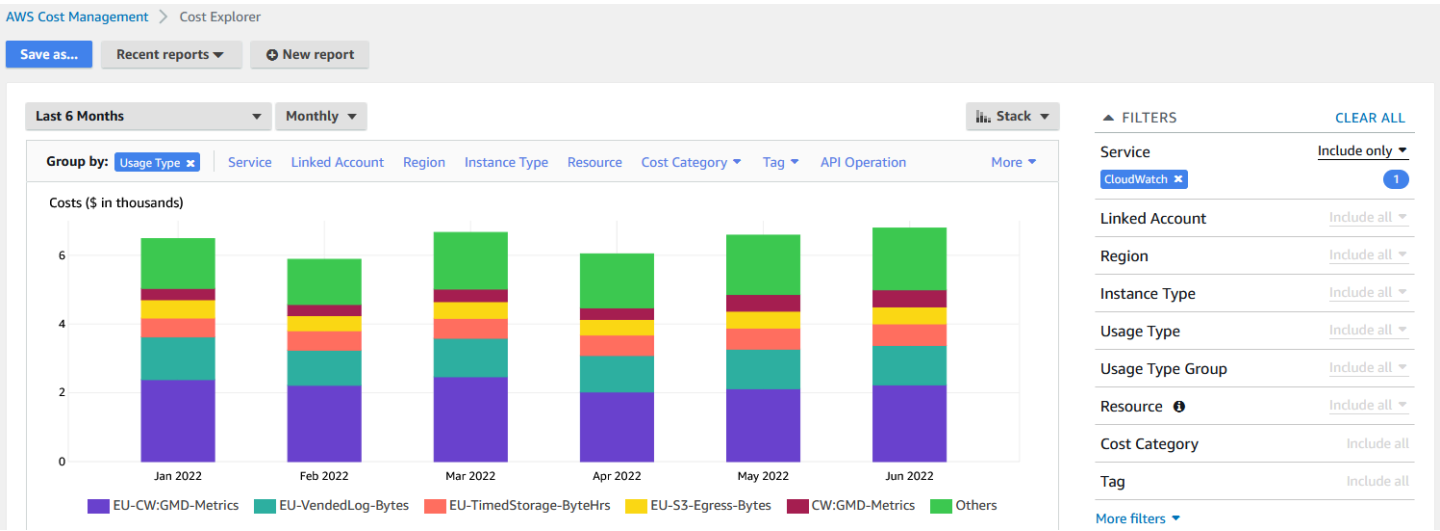
以下の手順では、Cost Explorer を使用して CloudWatch のコストと使用状況データを可視化および分析する方法について説明します。

CloudWatch のコストと使用状況データを可視化して分析するには

1. Cost Explorer コンソールにサインインします https://console.aws.amazon.com/cost-management/home#/custom。
2. [FILTERS] (フィルター) で、[Service] (サービス) に [CloudWatch] を選択します。
3. [Group by] (グループ化の条件) には [Usage Type] (使用タイプ) を選択します。また、次のような他のカテゴリ別に結果をグループ化することもできます。

   - [API Operation] (API オペレーション) – 最もコストがかかった API オペレーションを確認する。
   - [Region] (リージョン) – 最もコストがかかったリージョンを確認する。

次の画像は、CloudWatch の機能によって 6 か月間に発生したコストの例を示しています。



最もコストがかかった CloudWatch 機能を知るには、UsageType の値を確認します。例えば、EU-CW:GMD-Metrics は、CloudWatch バルク API リクエストによって発生したコストを表します。

> **Note**
>
> UsageType の文字列は、特定の機能およびリージョンと一致します。例えば、EU-CW:GMD-Metrics の最初の部分 (EU) は欧州 (アイルランド) リージョンと一致し、EU-CW:GMD-Metrics の後半の部分 (GMD-Metrics) は CloudWatch バルク API リクエストと一致します。
>
> UsageType の文字列全体は <Region>-CW:<Feature> または <Region>-<Feature> の形式になっています。
>
> 読みやすくするために、このドキュメント全体を通して、表の中の UsageType の文字列は、文字列の後半の部分に短縮されています。例えば、EU-CW:GMD-Metrics は GMD-Metrics に短縮されます。

次の表に、各 CloudWatch 機能の名前、各サブ機能の名前、および UsageType の文字列を示します。

CloudWatch 機能	CloudWatch サブ機能	**UsageType**
CloudWatch メトリクス	カスタムメトリクス	MetricMonitorUsage
	詳細モニターリング	

CloudWatch 機能	CloudWatch サブ機能	**UsageType**
		MetricMonitorUsage
	埋め込みメトリクス	MetricMonitorUsage
CloudWatch API リクエスト	API リクエスト	Requests
	バルク (取得)	GMD-Metrics
	Contributor Insights	GIRR-Metrics
	ビットマップイメージスナップショット	GMWI-Metrics
CloudWatch メトリクスストリーム	メトリクスストリーム	MetricStreamUsage
CloudWatch ダッシュボード	メトリクスが 50 以下のダッシュボード	DashboardsUsageHour-Basic
	メトリクスが 50 を超えるダッシュボード	DashboardsUsageHour
CloudWatch アラーム	標準 (メトリクスアラーム)	AlarmMonitorUsage
	高解像度 (メトリクスアラーム)	HighResAlarmMonitorUsage
	Metrics Insights クエリアラーム	MetricInsightAlarmUsage

CloudWatch 機能	CloudWatch サブ機能	**UsageType**
	複合 (集約アラーム)	CompositeAlarmMonitorUsage
CloudWatch Application Signals	Application Signals	Application-Signals
CloudWatch カスタムログ	収集 (取り込み)	DataProcessing-Bytes
	保存 (アーカイブ)	TimedStorage-ByteHrs
	分析 (クエリ)	DataScanned-Bytes
CloudWatch 低頻度アクセスログ	収集 (取り込み)	DataProcessingIA-Bytes
CloudWatch 出力ログ	配信 (Amazon CloudWatch Logs)	VendedLog-Bytes
	配信 (CloudWatch Logs 低頻度アクセスログ)	VendedLogIA-Bytes
	配信 (Amazon Simple Storage Service)	S3-Egress-ComprBytes S3-Egress-Bytes
	配信 (Amazon Data Firehose)	FH-Egress-Bytes
Contributor Insights	CloudWatch Logs (ルール)	ContributorInsightRules

CloudWatch 機能	CloudWatch サブ機能	**UsageType**
	CloudWatch Logs (イベント)	`ContributorInsightEvents`
	Amazon DynamoDB (ルール)	`ContributorRulesManaged`
	DynamoDB イベント)	`ContributorEventsManaged`
Canary (合成)	実行	`Canary-runs`
Evidently	イベント	`Evidently-event`
	分析ユニット	`Evidently-eau`
RUM	イベント	`RUM-event`

CloudWatch のコストと使用状況データを AWS Cost and Usage Report と Athena で分析する

CloudWatch のコストと使用状況データを分析するもう 1 つの方法は、AWS Cost and Usage Report を Amazon Athena とともに使用することです。AWS Cost and Usage Report には、コストと使用状況データが包括的にまとめて含まれています。コストと使用状況を追跡するレポートを作成し、これらのレポートを、任意の S3 バケットに発行できます。S3 バケットからレポートをダウンロードして削除することもできます。詳細については、「AWS Cost and Usage Report ユーザーガイド」の「AWS Cost and Usage Report とは?」を参照してください。

> **Note**
>
> AWS Cost and Usage Report の使用料は発生しません。ストレージに対して料金が発生するのは、Amazon Simple Storage Service (Amazon S3) にレポートを発行するときのみです。詳細については、「AWS Cost and Usage Report ユーザーガイド」の「クォータと制限」を参照してください。

Athena は、コストと使用状況データを分析するために AWS Cost and Usage Report で使用できるクエリサービスです。S3 バケット内のレポートは、最初にダウンロードしなくてもクエリできます。詳細については、「Amazon Athena ユーザーガイド」の「Amazon Athena とは」を参照してください。詳細については、「Amazon Athena ユーザーガイド」の「Amazon Athena とは」を参照してください。料金に関する詳細については、「Amazon Athena の料金」を参照してください。

以下の手順では、AWS Cost and Usage Report を有効にして、このサービスを Athena と統合するプロセスを説明します。この手順には、CloudWatch のコストと使用状況データを分析するために使用できる 2 つのクエリ例が含まれています。

> **Note**
>
> このドキュメント内のクエリ例は、どれでも使用してかまいません。このドキュメント内のクエリ例はすべて、costandusagereport という名前のデータベースに対応していて、2022 年 4 月の結果を表示します。この情報は、変更することができます。ただし、クエリを実行する前に、データベースの名前がクエリ内のデータベースの名前と一致していることを確認してください。

AWS Cost and Usage Report と Athena を使用してコストと使用状況のデータを分析するには

1. AWS Cost and Usage Report を有効にします。詳細については、「AWS Cost and Usage Report ユーザーガイド」の「コストと使用状況レポートを作成する」を参照してください。

   > **Tip**
   >
   > レポートを作成するときは、[Include resource IDs] (リソース ID を含める) を必ず選択してください。そうしないと、レポートに `line_item_resource_id` の列が含まれませ

ん。この行は、コストと使用状況データを分析するときにコストをさらに特定するのに役立ちます。

2. AWS Cost and Usage Report を Athena と統合します。詳細については、「AWS Cost and Usage Report ユーザーガイド」の「AWS CloudFormation テンプレートを使用した Athena の設定」を参照してください。
3. コストと使用状況のレポートをクエリします。

例: Athena クエリ

次のクエリを使用して、特定の月に最もコストが発生した CloudWatch 機能を表示できます。

```
SELECT
CASE
-- Metrics
WHEN line_item_usage_type LIKE '%%MetricMonitorUsage%%' THEN 'Metrics (Custom, Detailed
 monitoring management portal EMF)'
WHEN line_item_usage_type LIKE '%%Requests%%' THEN 'Metrics (API Requests)'
WHEN line_item_usage_type LIKE '%%GMD-Metrics%%' THEN 'Metrics (Bulk API Requests)'
WHEN line_item_usage_type LIKE '%%MetricStreamUsage%%' THEN 'Metric Streams'
-- Dashboard
WHEN line_item_usage_type LIKE '%%DashboardsUsageHour%%' THEN 'Dashboards'
-- Alarms
WHEN line_item_usage_type LIKE '%%AlarmMonitorUsage%%' THEN 'Alarms (Standard)'
WHEN line_item_usage_type LIKE '%%HighResAlarmMonitorUsage%%' THEN 'Alarms (High
 Resolution)'
WHEN line_item_usage_type LIKE '%%MetricInsightAlarmUsage%%' THEN 'Alarms (Metrics
 Insights)'
WHEN line_item_usage_type LIKE '%%CompositeAlarmMonitorUsage%%' THEN 'Alarms
 (Composite)'
-- Logs
WHEN line_item_usage_type LIKE '%%DataProcessing-Bytes%%' THEN 'Logs (Collect - Data
 Ingestion)'
-- Logs
WHEN line_item_usage_type LIKE '%%DataProcessingIA-Bytes%%' THEN 'Infrequent Access
 Logs (Collect - Data Ingestion)'
WHEN line_item_usage_type LIKE '%%TimedStorage-ByteHrs%%' THEN 'Logs (Storage -
 Archival)'
WHEN line_item_usage_type LIKE '%%DataScanned-Bytes%%' THEN 'Logs (Analyze - Logs
 Insights queries)'
-- Vended Logs
```

```
WHEN line_item_usage_type LIKE '%%VendedLog-Bytes%%' THEN 'Vended Logs (Delivered to
 CW)'
WHEN line_item_usage_type LIKE '%%VendedLogIA-Bytes%%' THEN 'Vended Infrequent Access
 Logs (Delivered to CW)'
WHEN line_item_usage_type LIKE '%%FH-Egress-Bytes%%' THEN 'Vended Logs (Delivered to
 Kinesis FH)'
WHEN (line_item_usage_type LIKE '%%S3-Egress-Bytes%%') OR (line_item_usage_type LIKE '%
%S3-Egress-
ComprBytes%%') THEN 'Vended Logs (Delivered to S3)'
-- Other
WHEN line_item_usage_type LIKE '%%Application-Signals%%' THEN 'Application Signals'
WHEN line_item_usage_type LIKE '%%Canary-runs%%' THEN 'Synthetics'
WHEN line_item_usage_type LIKE '%%Evidently%%' THEN 'Evidently'
WHEN line_item_usage_type LIKE '%%RUM-event%%' THEN 'RUM'
ELSE 'Others'
END AS UsageType,
-- REGEXP_EXTRACT(line_item_resource_id,'^(?:.+?:){5}(.+)$',1) as ResourceID,
-- SUM(CAST(line_item_usage_amount AS double)) AS UsageQuantity,
SUM(CAST(line_item_unblended_cost AS decimal(16,8))) AS TotalSpend
FROM
costandusagereport
WHERE
product_product_name = 'AmazonCloudWatch'
AND year='2022'
AND month='4'
AND line_item_line_item_type NOT IN
 ('Tax','Credit','Refund','EdpDiscount','Fee','RIFee')
-- AND line_item_usage_account_id = '123456789012' - If you want to filter on a
 specific account, you can
remove this comment at the beginning of the line and specify an AWS account.
GROUP BY
1
ORDER BY
TotalSpend DESC,
UsageType;
```

例: Athena クエリ

次のクエリを使用して、UsageType と Operation の結果を表示できます。これは、CloudWatch の機能でどのようにコストが発生したかを示しています。結果には UsageQuantity と TotalSpend の値も表示されます。これによって、総使用コストを確認できます。

> **Tip**
>
> UsageType の詳細を確認するには、次の行をこのクエリに追加します。
> line_item_line_item_description
> この行によって、[Description] (説明) という名前の列が作成されます。

```
SELECT
bill_payer_account_id as Payer,
line_item_usage_account_id as LinkedAccount,
line_item_usage_type AS UsageType,
line_item_operation AS Operation,
line_item_resource_id AS ResourceID,
SUM(CAST(line_item_usage_amount AS double)) AS UsageQuantity,
SUM(CAST(line_item_unblended_cost AS decimal(16,8))) AS TotalSpend
FROM
costandusagereport
WHERE
product_product_name = 'AmazonCloudWatch'
AND year='2022'
AND month='4'
AND line_item_line_item_type NOT IN
 ('Tax','Credit','Refund','EdpDiscount','Fee','RIFee')
GROUP BY
bill_payer_account_id,
line_item_usage_account_id,
line_item_usage_type,
line_item_resource_id,
line_item_operation
```

コスト削減と最適化のためのベストプラクティス

CloudWatch メトリクス

Amazon Elastic Compute Cloud (Amazon EC2)、Amazon S3、Amazon Data Firehose など多くの AWS のサービス は、メトリクスを CloudWatch に自動的に無料で送信します。ただし、次のカテゴリに分類されたメトリクスには、追加コストが発生する可能性があります。

- カスタムメトリクス、詳細モニターリング、埋め込みメトリクス
- API リクエスト

- メトリクスストリーム

詳細については、「Amazon CloudWatch メトリクスを使用する」を参照してください。

カスタムメトリクス、詳細モニターリング、埋め込みメトリクス

カスタムメトリクス

カスタムメトリクスを作成して、データポイントを任意の順序や比率で整理できます。

すべてのカスタムメトリクスは時間単位で比例配分されます。CloudWatch に送信されたときにのみ計測されます。メトリクスの料金の詳細については、「Amazon CloudWatch の料金」を参照してください。

次の表に、CloudWatch メトリクスに関連するサブ機能の名前を示します。表には、UsageType と Operation の文字列が含まれています。これは、メトリクス関連のコストを分析して特定するのに役立ちます。

> **Note**
>
> Athena でコストと使用状況のデータをクエリしているときに、次の表に示すメトリクスの詳細を取得するには、Operation の文字列を line_item_operation に示された結果と一致させます。

CloudWatch サブ機能	**UsageType**	**Operation**	目的
カスタムメトリクス	MetricMonitorUsage	MetricStorage	カスタムメトリクス
詳細モニターリング	MetricMonitorUsage	MetricStorage:AWS/*{Service}*	詳細モニターリング
埋め込みメトリクス	MetricMonitorUsage	MetricStorage:AWS/Logs-EMF	埋め込みメトリクスの口グを記録する

CloudWatch サブ機能	**UsageType**	**Operation**	目的
ログフィルター	`MetricMonitorUsage`	`MetricStorage:AWS/CloudWatchLogs`	ロググループメトリクスフィルター

詳細モニターリング

CloudWatch には、次の 2 つのタイプのモニターリングがあります。

- 基本モニターリング

  基本モニターリングは無料で、機能をサポートしているすべての AWS のサービス で自動的に有効になります。
- 詳細モニターリング

  詳細モニターリングにはコストがかかり、AWS のサービス によってさまざまな機能拡張が追加されています。詳細モニターリングをサポートしている AWS のサービス の場合は、そのサービスに対して有効にするかどうかを選択できます。詳細については、「基本モニターリングと詳細モニターリング」を参照してください。

> **Note**
>
> その他の AWS のサービス では、詳細モニターリングをサポートしていても、この機能を別の名前で参照している場合があります。例えば、Amazon S3 の詳細モニターリングはリクエストメトリクスと呼ばれます。

カスタムメトリクスと同様に、詳細モニターリングは時間単位で比例配分され、データが CloudWatch に送信されたときにのみ計測されます。詳細モニターリングでは、CloudWatch に送信されるメトクスの数によってコストが発生します。コストを削減するには、必要な場合にのみ詳細モニターリングを有効にします。詳細モニターリングの料金については、「Amazon CloudWatch の料金」を参照してください。

例: Athena クエリ

次のクエリを使用して、詳細モニターリングが有効になっている EC2 インスタンスを確認できます。

```
SELECT
bill_payer_account_id as Payer,
line_item_usage_account_id as LinkedAccount,
line_item_usage_type AS UsageType,
line_item_operation AS Operation,
line_item_resource_id AS ResourceID,
SUM(CAST(line_item_usage_amount AS double)) AS UsageQuantity,
SUM(CAST(line_item_unblended_cost AS decimal(16,8))) AS TotalSpend
FROM
costandusagereport
WHERE
product_product_name = 'AmazonCloudWatch'
AND year='2022'
AND month='4'
AND line_item_operation='MetricStorage:AWS/EC2'
AND line_item_line_item_type NOT IN
 ('Tax','Credit','Refund','EdpDiscount','Fee','RIFee')
GROUP BY
bill_payer_account_id,
line_item_usage_account_id,
line_item_usage_type,
line_item_resource_id,
line_item_operation,
line_item_line_item_description
ORDER BY line_item_operation
```

埋め込みメトリクス

CloudWatch 埋め込みメトリクス形式を使用すると、アプリケーションデータを口グデータとして取り込むことができるため、実用的なメトリクスを生成できます。詳細については、「高カーディナリティログの取り込みと CloudWatch 埋め込みメトリクスフォーマットによるメトリクスの生成」を参照してください。

埋め込みメトリクスでは、取り込まれた口グの数、アーカイブされた口グの数、生成されたカスタムメトリクスの数によってコストが発生します。

次の表に、CloudWatch 埋め込みメトリクス形式に関連するサブ機能の名前を示します。表には、UsageType と Operation の文字列が含まれています。これは、コストを分析して特定するのに役立ちます。

CloudWatch サブ機能	**UsageType**	**Operation**	目的
カスタムメトリクス	MetricMonitorUsage	MetricStorage:AWS/Logs-EMF	埋め込みメトリクスのログを記録する
ログの取り込み	DataProcessing-Bytes	PutLogEvents	一連のログイベントのバッチを指定されたロググループまたはログストリームにアップロードする
ログのアーカイブ	TimedStorage-ByteHrs	HourlyStorageMetering	CloudWatch Logs に 1 時間あたりのログとバイトあたりのログを保存する

コストを分析するには、AWS Cost and Usage Report を Athena とともに使用すると、コストを発生させているメトリクスを特定し、コストがどのように発生するかを判断できます。

CloudWatch 埋め込みメトリクス形式によって発生するコストを最大限に活用するには、カーディナリティの高いディメンションに基づくメトリクスの作成は避けてください。そうすれば、CloudWatch は一意のディメンションの組み合わせごとにカスタムメトリクスを作成しません。詳細については、「ディメンション」を参照してください。

埋め込みメトリクス形式を活用するために CloudWatch Container Insights を使用している場合は、メトリクス関連のコストを最大限に活用するための代替手段として AWS Distro for Open Telemetry を使用できます。Container Insights を使用して、コンテナ化されたアプリケーションとマイクロサービスのメトリクスとログを収集、集計、要約できます。Container Insights を有効にすると、CloudWatch エージェントはログを CloudWatch に送信し、ログを使用して埋め込みメトリクスを生成できるようにします。ただし、CloudWatch エージェントは CloudWatch に単に一定数のメトリクスを送信し、使用していないものも含め、使用可能なすべてのメトリクスに対して課金されます。AWS Distro for Open Telemetry では、CloudWatch に送信されるメトリクスとディメンションを設定およびカスタマイズできます。これにより、Container Insights が生成するデータ量とコストを削減できます。詳細については、以下のリソースを参照してください。

- Container Insights の使用

- AWS Distro for Open Telemetry

API リクエスト

CloudWatch には次のタイプの API リクエストがあります。

- API リクエスト
- バルク (取得)
- Contributor Insights
- ビットマップイメージスナップショット

API リクエストでは、リクエストタイプとリクエストされたメトリクスの数によってコストが発生します。

次の表に、API リクエストのタイプと、UsageType と Operation の文字列を示します。これは、API 関連のコストを分析して特定するのに役立ちます。

API リクエストのタイプ	**UsageType**	**Operation**	目的
API リクエスト	Requests	GetMetricStatistics	指定されたメトリクスの統計情報を取得する
	Requests	ListMetrics	指定されたメトリクスを一覧表示する
	Requests	PutMetricData	メトリックスのデータポイントを CloudWatch に発行する
	Requests	GetDashboard	指定されたダッシュボードの詳細を表示する

API リクエストのタイプ	**UsageType**	**Operation**	目的
	Requests	ListDashboards	アカウントのダッシュボードを一覧表示する
	Requests	PutDashboard	ダッシュボードを作成または更新する
	Requests	DeleteDashboards	指定されたダッシュボードをすべて削除する
バルク (取得)	GMD-Metrics	GetMetricData	CloudWatch メトリクス値を取得する
Contributor Insights	GIRR-Metrics	GetInsightRuleReport	Contributor Insights ルールによって収集された時系列データを返す
ビットマップイメージスナップショット	GMWI-Metrics	GetMetricWidgetImage	1 つまたは複数の CloudWatch メトリクスのスナップショットをビットマップイメージとして取得する

コストを分析するには、Cost Explorer を使用して、結果を API オペレーションによってグループ化します。

API リクエストのコストはさまざまで、AWS 無料利用枠の制限以下で提供される API 呼び出しの数を超えるとコストが発生します。

> **Note**
>
> GetMetricData と GetMetricWidgetImage は、AWS 無料利用枠の制限には含まれません。詳細については、「AWS Billing ユーザーガイド」の「AWS 無料利用枠の使用」を参照してください。

一般的にコストがかかる API リクエストは、Put と Get リクエストです。

PutMetricData

PutMetricData が呼び出されるたびにコストが発生し、ユースケースによっては多大なコストが発生する可能性があります。詳細については、「Amazon CloudWatch API リファレンス」の「PutMetricData」を参照してください。

PutMetricData によって発生するコストを最大限に活用するには、より多くのデータを一括して API 呼び出しをします。ユースケースによっては、CloudWatch Logs または CloudWatch 埋め込みメトリクス形式を使用して、メトリクスデータを盛り込むことを検討してください。詳細については、以下のリソースを参照してください。

- Amazon CloudWatch Logs ユーザーガイドの「Amazon CloudWatch Logs とは」
- 高カーディナリティログの取り込みと CloudWatch 埋め込みメトリクスフォーマットによるメトリクスの生成
- Amazon CloudWatch 埋め込みカスタムメトリクスでコストを削減し、お客様に重点を置く

GetMetricData

また、GetMetricData でも多大なコストが発生する可能性があります。コストを押し上げる一般的なユースケースには、データを取得してインサイトを生成するサードパーティのモニターリングツールが関連しています。詳細については、「Amazon CloudWatch API リファレンス」の「GetMetricData」を参照してください。

GetMetricData によって発生するコストを削減するには、モニターリングし使用するデータのみを取得することを検討するか、データを取得する頻度を減らすことを検討してください。ユースケースによっては、GetMetricData ではなくメトリクスストリームを使用することを検討するとよいかもしれません。これにより、低コストでデータをほぼリアルタイムでサードパーティにプッシュできます。詳細については、以下のリソースを参照してください。

- メトリクスストリームの使用

- CloudWatch メトリクスストリーム - AWS メトリクスをパートナーとアプリケーションにリアルタイムで送信する

GetMetricStatistics

ユースケースによっては、GetMetricData ではなく GetMetricStatistics を使用することを検討するとよいかもしれません。GetMetricData を使用すると、データを迅速かつ大規模に取得できます。しかし、GetMetricStatistics は最大 100 万回の API リクエストに対する AWS 無料利用枠の制限に含まれます。1 回の呼び出しでそれほど多くのメトリクスとデータポイントを取得する必要がない場合に、コストを削減できます。詳細については、以下のリソースを参照してください。

- 「Amazon CloudWatch API リファレンス」の「GetMetricStatistics」
- GetMetricData を使うべきでしょうか、それとも GetMetricStatistics を使うべきでしょうか。

> **Note**
>
> 外部呼び出し元は API 呼び出しを行います。現在、これらの発信者を特定する唯一の方法は、CloudWatch チームへのテクニカルサポートリクエストをオープンし、その発信者に関する情報を求めることです。テクニカルサポートリクエストの作成の詳細については、「AWS の技術サポートを受けるにはどうすればよいですか?」を参照してください。

CloudWatch メトリクスストリーム

CloudWatch メトリクスストリームを使用すると、メトリクスを継続的に AWS の送信先とサードパーティのサービスプロバイダーの送信先に送信できます。

メトリクスストリームは、メトリクス更新の件数によってコストが発生します。メトリクス更新には、常に次の統計情報の値が含まれています。

- `Minimum`
- `Maximum`
- `Sample Count`
- `Sum`

詳細については、「ストリーミングできる統計情報」を参照してください。

CloudWatch メトリクスストリームによって発生するコストを分析するには、AWS Cost and Usage Report を Athena とともに使用します。これにより、コストが発生しているメトリクストリームを特定し、コストがどのように発生するかを判断できます。

例: Athena クエリ

次のクエリを使用して、Amazon リソースネーム (ARN) ごとにコストが発生するメトリクスストリームを追跡できます。

```
SELECT
SPLIT_PART(line_item_resource_id,'/',2) AS "Stream Name",
line_item_resource_id as ARN,
SUM(CAST(line_item_unblended_cost AS decimal(16,2))) AS TotalSpend
FROM
costandusagereport
WHERE
product_product_name = 'AmazonCloudWatch'
AND year='2022'
AND month='4'
AND line_item_line_item_type NOT IN
 ('Tax','Credit','Refund','EdpDiscount','Fee','RIFee')
-- AND line_item_usage_account_id = '123456789012' - If you want to filter on a
 specific account, you can
remove this comment at the beginning of the line and specify an AWS account.
AND line_item_usage_type LIKE '%%MetricStreamUsage%%'
GROUP BY line_item_resource_id
ORDER BY TotalSpend DESC
```

CloudWatch メトリクスストリームによって発生するコストを削減するには、ビジネス価値をもたらすメトリクスのみをストリーミングします。また、使用していないメトリクスストリームを停止または一時停止することもできます。

CloudWatch アラーム

CloudWatch アラームでは、1 つのメトリクスに基づくアラーム、Metrics Insights のクエリに基づくアラーム、他のアラームを監視する複合アラームを作成できます。

> **Note**
>
> メトリクスアラームおよび複合アラームのコストは、1 時間単位で比例配分されます。アラームのコストが発生するのは、アラームが存在している間のみです。コストを最適化するためには、設定ミスのあるアラームや価値の低いアラームを残さないことが必要です。

これを解決するため、不要になった CloudWatch アラームのクリーンアップを自動化できます。詳細については、「Amazon CloudWatch アラームの大規模なクリーンアップを自動化する」を参照してください。

[Metric alarms] (メトリクスアラーム)

メトリクスアラームには次の解像度設定があります。

- 標準 (60 秒ごとに評価)
- 高解像度 (10 秒ごとに評価)

メトリクスアラームを作成する場合、コストはアラームの解像度の設定と、アラームが参照するメトリクスの数に基づきます。たとえば、1 つのメトリクスを参照するメトリクスアラームには、1 時間あたり 1 つのアラームメトリックのコストが発生します。詳細については、「Using Amazon CloudWatch alarms」(Amazon CloudWatch アラームを使用する) を参照してください。

複数のメトリクスを参照するメトリクス数式を含むメトリクスアラームを作成する場合、メトリクスの数式で参照されるアラームメトリクスごとにコストが発生します。メトリクス数式を含むメトリクスアラームを作成する方法については、「メトリクスの数式に基づく CloudWatch アラームの作成」を参照してください。

アラームが過去のメトリクスデータを分析して期待値のモデルを作成する異常検出アラームを作成する場合、アラームで参照される各アラームメトリクスと 2 つの追加メトリクス (異常検出モデルが作成する上限および下限帯域メトリクス用に 1 つ) のコストが発生します。異常検出アラームを作成する方法については、「異常検出に基づく CloudWatch アラームの作成」を参照してください。

Metrics Insights クエリアラーム

Metric Insights クエリアラームは特定のタイプのメトリクスアラームで、標準解像度 (60 秒ごとに評価) でのみ使用できます。

Metric Insights クエリアラームを作成する場合、コストはアラームが参照するクエリによって分析されるメトリクスの数に基づきます。例えば、フィルターが 10 個のメトリクスに一致するクエリを参照する Metric Insights クエリアラームでは、1 時間あたりメトリクス 10 個分の分析コストが発生します。詳細については、「Amazon CloudWatch の料金」で料金の例を確認してください。

Metrics Insights クエリとメトリクスの数式の両方を含むアラームを作成する場合、Metrics Insights クエリアラームとして報告されます。Metrics Insights クエリで分析されるメトリクスに加え、他の

メトリクスを参照するメトリクスの数式がアラームに含まれる場合、メトリクスの数式で参照されるアラームメトリクスごとにコストが発生します。メトリクス数式を含むメトリクスアラームを作成する方法については、「メトリクスの数式に基づく CloudWatch アラームの作成」を参照してください。

[Composite alarms] (複合アラーム)

複合アラームには、他のアラームの状態を評価して自身の状態を判断する方法を指定するルール式が含まれています。複合アラームは、評価される他のアラームの数に関係なく、1 時間あたりの標準コストが発生します。複合アラームがルール式で参照するアラームには、個別のコストが発生します。詳細については、「複合アラームの作成」を参照してください。

[Alarm usage types] (アラームの使用タイプ)

次の表に、CloudWatch アラームに関連するサブ機能の名前を示します。表には、UsageType の文字列が含まれています。これは、アラーム関連のコストを分析して特定するのに役立ちます。

CloudWatch サブ機能	**UsageType**
標準メトリクスアラーム	`AlarmMonitorUsage`
高解像度メトリクスアラーム	`HighResAlarmMonitorUsage`
Metrics Insights クエリアラーム	`MetricInsightAlarmUsage`
複合アラーム	`CompositeAlarmMonitorUsage`

アラームコストの削減

4 つ以上のメトリクスを集計するメトリクス演算アラームによって発生するコストを最大限に活用するために、データが CloudWatch に送信される前にデータを集計するカスタムメトリクスを作成できます。こうすると、複数のメトリクスのデータを集計するアラームではなく、単一のメトリクスのアラームを作成できます。詳細については、カスタムメトリクスの発行を参照してください。

Metrics Insights クエリアラームによって発生するコストを最大限に活用するには、クエリに使用されるフィルターがモニターリング対象のメトリクスにのみ一致するようにします。

コストを削減する最善の方法は、不要なアラームや使用していないアラームをすべて削除することです。たとえば、すでに存在しない AWS リソースによって発行されたメトリクスを評価するアラームを削除できます。

Example: Check for alarms in **`INSUFFICIENT_DATA`** state with **`DescribeAlarms`**

リソースを削除しても、そのリソースが発行するメトリックアラームを削除しない場合、アラームは引き続き存在し、通常 `INSUFFICIENT_DATA` の状態になります。アラームが `INSUFFICIENT_DATA` の状態になっているかどうかを確認するには、次の AWS Command Line Interface (AWS CLI) コマンドを使用します。

```
$ aws cloudwatch describe-alarms –state-value INSUFFICIENT_DATA
```

その他に次のようなコスト削減方法があります。

- 正しいメトリクスのアラームを作成していることを確認してください。
- 作業していないリージョンでアラームが有効になっていないことを確認してください。
- 複合アラームはノイズを低減しますが、追加コストも発生することに留意してください。
- 標準アラームと高解像度アラームのどちらを作成するかを決めるときは、ユースケースと、各タイプのアラームがもたらす価値を考慮してください。

CloudWatch ログ

Amazon CloudWatch Logs には次のログタイプがあります。

- カスタムログ (アプリケーション用に作成するログ)
- 提供されるログ (Amazon Virtual Private Cloud (Amazon VPC) や Amazon Route 53 など、他の AWS のサービス がユーザーに代わって作成するログ)

提供されるログの詳細については、「Amazon CloudWatch Logs ユーザーガイド」の「特定の AWS のサービスからのログの記録を有効にする」を参照してください。

カスタムログと提供されるログでは、収集、保存、分析されたログの数に基づいてコストが発生します。これとは別に、提供されるログでは Amazon S3 と Firehose への配信コストが発生します。

次の表に、CloudWatch Logs 機能の名前と関連するサブ機能の名前を示します。表には、`UsageType` と `Operation` の文字列が含まれます。これは、ログ関連のコストを分析して特定するのに役立ちます。

CloudWatch Logs の機能	CloudWatch Logs のサブ機能	**UsageType**	**Operation**	目的
カスタムログ	収集 (取り込み)	DataProcessing-Bytes	PutLogEvents	ログのバッチを特定のログストリームにアップロードする
	保存 (アーカイブ)	TimedStorage-ByteHrs	HourlyStorageMetering	CloudWatch Logs に 1 時間あたりのログとバイトあたりのログを保存する
	分析 (Logs Insights クエリ)	DataScanned-Bytes	StartQuery	CloudWatch Logs Insights クエリによってスキャンされたデータをログに記録する
提供されるログ	配信 (CloudWatch Logs)	VendedLog-Bytes	PutLogEvents	ログのバッチを特定のログストリームにアップロードする
	配信 (Amazon S3)	S3-Egress-ComprBytes S3-Egress-Bytes	LogDelivery	提供されるログを送信する (CloudWatch、Amazon S3、または Firehose)
	配信 (Firehose)	FH-Egress-Bytes	LogDelivery	提供されるログを送信する (CloudWatch、Amazon

CloudWatch Logs の機能	CloudWatch Logs のサブ機能	**UsageType**	**Operation**	目的
				S3、または Firehose)

コストを分析するには、AWS Cost and Usage Report を Athena とともに使用して、コストを発生させているログを特定し、コストがどのように発生しているかを判断できます。

例: Athena クエリ

次のクエリを使用して、リソース ID 別にコストが発生するログを追跡できます。

```
SELECT
bill_payer_account_id as Payer,
line_item_usage_account_id as LinkedAccount,
line_item_resource_id AS ResourceID,
line_item_usage_type AS UsageType,
SUM(CAST(line_item_unblended_cost AS decimal(16,8))) AS TotalSpend,
SUM(CAST(line_item_usage_amount AS double)) AS UsageQuantity
FROM
costandusagereport
WHERE
product_product_name = 'AmazonCloudWatch'
AND year='2022'
AND month='4'
AND line_item_operation IN
 ('PutLogEvents','HourlyStorageMetering','StartQuery','LogDelivery')
AND line_item_line_item_type NOT IN
 ('Tax','Credit','Refund','EdpDiscount','Fee','RIFee')
GROUP BY
bill_payer_account_id,
line_item_usage_account_id,
line_item_usage_type,
line_item_resource_id,
line_item_operation
ORDER BY
TotalSpend DESC
```

CloudWatch Logs によって発生するコストを最大限に活用するには、以下を考慮してください。

- ビジネス価値をもたらすイベントのみをログに記録します。これにより、取り込みコストが削減されます。
- ログの保持設定を変更して、ストレージにかかるコストを抑えます。詳細については、「Amazon CloudWatch Logs User Guide」(Amazon CloudWatch Logs ユーザーガイド) の「Change log data retention in CloudWatch Logs」(CloudWatch ログでのログデータ保管期間の変更) を参照してください。
- CloudWatch Logs Insights が自動的に履歴に保存するクエリを実行します。これにより、分析にかかるコストが少なくなります。詳細については、「Amazon CloudWatch Logs ユーザーガイド」の「実行中のクエリまたはクエリ履歴を表示する」を参照してください。
- CloudWatch エージェントを使用して、システムログとアプリケーションログを収集し、CloudWatch に送信します。これにより、条件を満たすログイベントのみを収集できます。詳細については、「Amazon CloudWatch エージェントは、ログフィルター式のサポートを追加」を参照してください。

提供されるログのコストを削減するには、ユースケースを検討し、ログを CloudWatch または Amazon S3 に送信するかどうかを決定します。詳細については、「Amazon CloudWatch Logs ユーザーガイド」の「Amazon S3 に送信されたログ」を参照してください。

> **Tip**
>
> メトリクスフィルター、サブスクリプションフィルター、CloudWatch Logs Insights、Contributor Insights を使用する場合は、公開ログを CloudWatch に送信します。また、VPC フローログを操作し、監査とコンプライアンスの目的で使用している場合は、提供されるログを Amazon S3 に送信します。
>
> VPC フローログを S3 バケットに公開することで発生する料金を追跡する方法については、「AWS Cost and Usage Report とコスト配分タグを使用して、Amazon S3 への VPC フローログのデータインジェストのコストを知る」を参照してください。

CloudWatch Logs によって発生するコストを最大限に活用する方法の詳細については、「CloudWatch Logs の請求が急に増加したのですが、どのロググループが原因でしょうか?」を参照してください。

Amazon CloudWatch ダッシュボードの使用

Amazon CloudWatch ダッシュボードは、CloudWatch コンソールにあるカスタマイズ可能なホームページであり、ダッシュボードを使用すれば、異なるリージョンにまたがっているリソースでも、1 つのビューでモニターリングできます。CloudWatch ダッシュボードを使用して、AWS リソースのメトリクスおよびアラームをカスタマイズした状態で表示することができます。

ダッシュボードでは、以下を作成することが可能です。

- 選択されたメトリクスとアラームのための単一ビュー。これは、1 つ、または複数のリージョン全体のリソースとアプリケーションの正常性を評価するために役立ちます。各グラフのメトリクスごとに使用する色を選択できるため、複数のグラフにわたる同一のメトリクスを追跡しやすくなります。
- 操作イベント中の特定の問題への対処法に関して、チームのメンバーにガイダンスを提供する運営計画。
- 重要なリソースとアプリケーション測定の共通表示。これをチームメンバー間で共有して操作イベント中のコミュニケーションフローを円滑化できます。

複数の AWS アカウントをお持ちの場合は、CloudWatch クロスアカウントオブザーバビリティを設定してから、モニタリングアカウントに高度なクロスアカウントダッシュボードを作成します。これらのダッシュボードには、ソースアカウントのメトリクスのグラフや、ソースアカウントのロググループのクエリを含む CloudWatch Logs Insights ウィジェットを含めることができます。さらに、モニタリングアカウントで作成したアラームで、ソースアカウントのメトリクスを監視できます。詳細については、「CloudWatch のクロスアカウントオブザーバビリティ」を参照してください。

ダッシュボードは、コンソールから作成するか、AWS CLI または `PutDashboard` API オペレーションを使用して作成できます。ダッシュボードをお気に入りリストに追加することで、お気に入りのダッシュボードだけでなく、最近アクセスしたダッシュボードにもアクセスできます。詳細については、「[Favorites] (お気に入り) リストにダッシュボードを追加する」を参照してください。

CloudWatch ダッシュボードにアクセスするには、次のいずれかが必要です。

- `AdministratorAccess` ポリシー
- `CloudWatchFullAccess` ポリシー
- これらの特定のアクセス許可の 1 つ以上を含むカスタムポリシー。
  - ダッシュボードを表示できるようにするには `cloudwatch:GetDashboard` と `cloudwatch:ListDashboards`。

- ダッシュボードを作成または変更できるようにするには `cloudwatch:PutDashboard`。
- `cloudwatch:DeleteDashboards`ダッシュボードを削除できるようにするには 。

目次

CloudWatch ダッシュボードの作成

開始するには、CloudWatch ダッシュボードを作成します。複数のダッシュボードを作成して、ダッシュボードをお気に入りリストに追加できます。AWS アカウント に作成できるダッシュボードの数に制限はありません。すべてのダッシュボードはグローバルです。ダッシュボードはリージョン固有ではありません。

次の手順は、CloudWatch コンソールからダッシュボードを作成する方法を説明しています。`PutDashboard` API オペレーションを使用して、コマンドラインインターフェイスからダッシュボードを作成できます。API オペレーションには、ダッシュボードコンテンツを定義する JSON 文字列が含まれています。`PutDashboard` API オペレーションを使用するダッシュボード作成の詳細については、「Amazon CloudWatch API Reference」(Amazon CloudWatch API リファレンス) の「PutDashboard」を参照してください。

> **Tip**
>
> PutDashboard API オペレーションを使用して新しいダッシュボードを作成する場合、すでに存在するダッシュボードの JSON 文字列を使用できます。

コンソールからダッシュボードを作成するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで [Dashboards] (ダッシュボード)、[Create dashboard] (ダッシュボードの作成) の順に選択します。
3. [Create new dashboard] (新しいダッシュボードの作成) ダイアログボックスで、ダッシュボード名を入力し、[Create dashboard] (ダッシュボードの作成) をクリックします。

   [CloudWatch-Default] 名または [CloudWatch-Default-*ResourceGroupName*] 名を使用する場合、[Default Dashboard] (デフォルトのダッシュボード) の CloudWatch ホームページの概要にダッシュボードが表示されます。(詳細については、Amazon CloudWatch の開始方法 を参照してください)。
4. [Add to this dashboard] (これをダッシュボードに追加) ダイアログボックスで、次のいずれかを実行します。
   - ダッシュボードにグラフを追加するには、[Line] (線) または [Stacked area] (スタックされたエリア) を選択し、[Configure] (設定) を選択します。[Add metric graph] (メトリクスグラフの追加) ダイアログボックスで、グラフ化するメトリクスを選択し、[Create widget] (ウィジェットの作成) をクリックします。14 日を超える期間にわたってデータを発行していないため、メトリクスがダイアログボックスに表示されない場合は、手動で追加できます。(詳細については、CloudWatch ダッシュボードで手動でメトリクスをグラフ化する を参照してください)。
   - メトリクスを表示する数をダッシュボードに追加するには、[Number] (数値)、[Configure] (設定) の順にクリックします。[Add metric graph] (メトリクスグラフの追加) ダイアログボックスで、グラフ化するメトリクスを選択し、[Create widget] (ウィジェットの作成) をクリックします。
   - ダッシュボードにテキストブロックを追加するには、[Text] (テキスト)、[Configure] (設定) の順にクリックします。[New text widget] (新しいテキストウィジェット) ダイアログボックスの[Markdown] (マークダウン) で、マークダウンを使用するテキストをフォーマットして、[Create widget] (ウィジェットの作成) をクリックします。

5. (オプション)、[Add widget] (ウィジェットの追加) を選択してステップ 4 を繰り返し、別のウィジェットをダッシュボードに追加します。このステップは複数回繰り返すことができます。

   ダッシュボードの各グラフの右上には、[情報] のアイコンがあります。このアイコンをクリックすると、グラフ内のメトリクスの説明が表示されます。
6. [ダッシュボードの保存] を選択します。

CloudWatch クロスアカウントオブザーバビリティダッシュボード

複数の AWS アカウントをお持ちの場合は、CloudWatch クロスアカウントオブザーバビリティを設定してから、モニターリングアカウントに高度なクロスアカウントダッシュボードを作成します。アカウントの境界なしに、メトリクス、ログ、トレースをシームレスに検索、可視化、分析できます。

CloudWatch クロスアカウントオブザーバビリティの設定の詳細については、「CloudWatch のクロスアカウントオブザーバビリティ」を参照してください。

CloudWatch クロスアカウントオブザーバビリティでは、モニタリングアカウントのダッシュボードで次の操作を行うことができます。

- ソースアカウントにあるメトリクスのグラフを検索、表示、作成する。1 つのグラフに複数のアカウントのメトリクスを含めることができます。
- ソースアカウントのメトリクスを監視するアラームをモニタリングアカウントに作成する。
- ソースアカウントにあるロググループのログイベントを表示し、ソースアカウントのロググループの CloudWatch Logs Insights クエリを実行する。モニタリングアカウントの CloudWatch Logs Insights クエリを 1 回実行すると、複数のソースアカウントの複数のロググループに同時にクエリを実行できます。
- X-Ray のトレースマップでソースアカウントのノードを表示します。その後、マップを特定のソースアカウントにフィルタリングできます。

モニタリングアカウントにサインインすると、CloudWatch クロスアカウントオブザーバビリティ機能をサポートするすべてのページの右上に青色の [Monitoring account] (モニタリングアカウント) バッジが表示されます。

クロスアカウントクロスリージョンダッシュボード

クロスアカウントクロスリージョンダッシュボードを作成して、複数の AWS アカウントおよび複数のリージョンの CloudWatch データを 1 つのダッシュボードにまとめることができます。この全体

的なダッシュボードから、アプリケーション全体のビューを取得することができ、アカウントのサインインやログアウトを行ったり、リージョンを切り替えたりしなくてもより具体的なダッシュボードにドリルダウンすることもできます。

クロスアカウントクロスリージョンダッシュボードは、AWS Management Consoleやプログラムにより作成できます。

前提条件

クロスアカウントクロスリージョンダッシュボードを作成する前に、少なくとも 1 つの共有アカウントと少なくとも 1 つのモニターリングアカウントを有効にする必要があります。さらに、CloudWatch コンソールを使用してクロスアカウントダッシュボードを作成するには、クロスアカウント機能に対してコンソールを有効にする必要があります。詳細については、「クロスアカウントクロスリージョン CloudWatch コンソール」を参照してください。

AWS Management Console でのクロスアカウントクロスリージョンダッシュボードの作成と使用

AWS Management Consoleを使用して、クロスアカウントクロスリージョンダッシュボードを作成できます。

クロスアカウントクロスリージョンダッシュボードを作成するには

1. モニターリングアカウントにサインインします。
2. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
3. ナビゲーションペインで、ダッシュボードを選択します。
4. ダッシュボードを選択するか、新しいダッシュボードを作成します。
5. 画面の上部で、アカウントとリージョンを切り替えることができます。ダッシュボードを作成するときに、複数のアカウントとリージョンのウィジェットを含めることができます。ウィジェットには、グラフ、アラーム、CloudWatch Logs Insights ウィジェットが含まれます。

さまざまなアカウントとリージョンのメトリクスを使用したグラフの作成

1. モニターリングアカウントにサインインします。
2. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
3. ナビゲーションペインで、[Metrics] (メトリクス)、[All metrics] (すべてのメトリクス) の順に選択します。

4. メトリクスの追加元のアカウントとリージョンを選択します。アカウントとリージョンは、画面の右上にあるアカウントとリージョンのドロップダウンメニューから選択できます。
5. 必要なメトリクスをグラフを追加します。詳細については、「メトリクスのグラフ化」を参照してください。
6. ステップ 4 ~ 5 を繰り返して、他のアカウントとリージョンからメトリクスを追加します。
7. (オプション) [Graphed metrics] タブを選択し、選択したメトリクスを使用するメトリクス数学関数を追加します。詳細については、「Metric Math を使用する」を参照してください。

   1 つのグラフを設定して、複数の SEARCH 関数を含めることもできます。各検索では、異なるアカウントまたはリージョンを参照できます。
8. グラフの作成が完了したら、[アクション]、[ダッシュボードに追加] の順に選択します。

   クロスアカウントダッシュボードを選択し、[ダッシュボードに追加] を選択します。

クロスアカウントダッシュボードへの別のアカウントのアラームの追加

1. モニターリングアカウントにサインインします。
2. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
3. ページの上部で、アラームが配置されているアカウントを選択します。
4. ナビゲーションペインで、[Alarms] を選択します。
5. 追加するアラームの横にあるチェックボックスをオンにし、[ダッシュボードに追加] を選択します。
6. 追加するクロスアカウントダッシュボードを選択し、[ダッシュボードに追加] を選択します。

プログラムを使用してクロスアカウントクロスリージョンダッシュボードを作成する

AWS API と SDK を使用して、プログラムによりダッシュボードを作成できます。詳細については、「PutDashboard」を参照してください。

クロスアカウントクロスリージョンダッシュボードを有効にするため、次の表と例に示すように、ダッシュボードの本文構造に新しいパラメータを追加しました。ダッシュボード本体構造全体の詳細については、「ダッシュボードの本文構造と構文」を参照してください。

Parameter	以下を使用	スコープ	デフォルト
accountId	ウィジェットまたはメトリクスが配置されているアカウントの ID を指定します。	ウィジェットまたはメトリクス	現在ログインしているアカウント
region	メトリクスのリージョンを指定します。	ウィジェットまたはメトリクス	コンソールで選択されている現在のリージョン

次の例は、クロスアカウントクロスリージョンダッシュボードのウィジェットの JSON ソースを示しています。

この例では、ウィジェットレベルで accountId フィールドを共有アカウントの ID に設定します。これにより、このウィジェットのすべてのメトリクスをその共有アカウントとリージョンから取得するよう指定されます。

```
{
    "widgets": [
        {
            ...
            "properties": {
              "metrics": [
                     …..
                     ],
                    "accountId": "111122223333",
                    "region": "us-east-1"
           }
       }
  ]
}
```

この例では、各メトリクスのレベルごとに異なる accountId フィールドを設定します。この例では、このメトリクスの数学式のさまざまなメトリクスが、さまざまな共有アカウントとさまざまなリージョンから取得されています。

```
{
"widgets": [
    {
```

```
        ...
        "properties": {
            "metrics": [
                [ { "expression": "SUM(METRICS())", "label": "[avg: ${AVG}]
 Expression1", "id": "e1", "stat": "Sum" } ],
                [ "AWS/EC2", "CPUUtilization", { "id": "m2", "accountId":
 "5555666677778888", "region": "us-east-1", "label": "[avg: ${AVG}] ApplicationALabel
 " } ],
                [ ".", ".", { "id": "m1", "accountId": "9999000011112222", "region":
 "eu-west-1", "label": "[avg: ${AVG}] ApplicationBLabel" } ]
            ],
            "view": "timeSeries",
            "region":"us-east-1", ---> home region of the metric. Not present in above
 example
            "stacked": false,
            "stat": "Sum",
            "period": 300,
            "title": "Cross account example"
        }
    }
]
}
```

この例は、アラームウィジェットを示しています。

```
{
    "type": "metric",
    "x": 6,
    "y": 0,
    "width": 6,
    "height": 6,
    "properties": {
        "accountID": "111122223333",
        "title": "over50",
        "annotations": {
            "alarms": [
                "arn:aws:cloudwatch:us-east-1:379642911888:alarm:over50"
            ]
        },
        "view": "timeSeries",
        "stacked": false
    }
}
```

この例は、CloudWatch Logs Insights ウィジェットの場合です。

```
{
    "type": "log",
    "x": 0,
    "y": 6,
    "width": 24,
    "height": 6,
    "properties": {
        "query": "SOURCE 'route53test' | fields @timestamp, @message\n| sort @timestamp
 desc\n| limit 20",
        "accountId": "111122223333",
        "region": "us-east-1",
        "stacked": false,
        "view": "table"
    }
}
```

プログラムによりダッシュボードを作成するもう 1 つの方法として、最初にAWS Management Consoleでダッシュボードを作成し、次にこのダッシュボードの JSON ソースをコピーできます。これを行うには、ダッシュボードをロードし、[アクション]、[ソースの表示/編集] の順に選択します。その後、このダッシュボード JSON をコピーして、同様のダッシュボードを作成するためのテンプレートとして使用できます。

ダッシュボード変数を使用して柔軟なダッシュボードを作成する

ダッシュボード変数を使用して、ダッシュボード内の入力フィールドの値に応じて、複数のウィジェットでさまざまなコンテンツをすばやく表示できる柔軟なダッシュボードを作成します。例えば、さまざまな Lambda 関数または Amazon EC2 インスタンス ID をすばやく切り替えることができるダッシュボードや、さまざまな AWS リージョンに切り替えることができるダッシュボードを作成できます。

変数を使用するダッシュボードを作成した後、同じ変数パターンを他の既存のダッシュボードにコピーできます。

ダッシュボード変数を使用すると、ダッシュボードを使用するユーザーの運用ワークフローが改善されます。また、同様のダッシュボードを複数作成するのではなく、1 つのダッシュボードでダッシュボード変数を使用するため、コストも削減できます。

> **Note**
>
> ダッシュボード変数を含むダッシュボードを共有する場合、共有先のユーザーは変数の値を変更できません。

ダッシュボード変数の種類

ダッシュボード変数は、プロパティ変数またはパターン変数にすることができます。

- プロパティ変数は、ダッシュボードにおけるすべてのウィジェット内のプロパティのすべてのインスタンスを変更します。このプロパティは、ダッシュボードの JSON ソース内の任意の JSON プロパティ(`region` など) にすることができます。または、`InstanceID` や `FunctionName` などのメトリクスのディメンション名にすることもできます。

  プロパティ変数を使用するチュートリアルについては、「チュートリアル: 関数名を変数として使用して Lambda ダッシュボードを作成する」を参照してください。

  ダッシュボードの JSON ソースの詳細については、「ダッシュボード本体の構造と構文」を参照してください。CloudWatch コンソールで、[アクション]、[ソースを表示/編集] を選択すると、カスタムダッシュボードの JSON ソースを表示できます。
- パターン変数は、正規表現パターンを使用して、JSON プロパティのすべて、またはその特定の部分のみを変更します。

  パターン変数を使用するチュートリアルについては、「チュートリアル: 正規表現パターンを使用してリージョンを切り替えるダッシュボードを作成する」を参照してください。

プロパティ変数はほとんどのユースケースに適用され、設定はそれほど複雑ではありません。

トピック

- チュートリアル: 関数名を変数として使用して Lambda ダッシュボードを作成する
- チュートリアル: 正規表現パターンを使用してリージョンを切り替えるダッシュボードを作成する
- 変数を別のダッシュボードにコピーする

チュートリアル: 関数名を変数として使用して Lambda ダッシュボードを作成する

この手順のステップでは、プロパティ変数を使用して、さまざまなメトリクスグラフを表示する柔軟なダッシュボードを作成する方法を示します。これには、異なる Lambda 関数間ですべてのグラフのメトリクスを切り替えるために使用できるダッシュボード上のドロップダウン選択ボックスが含まれます。

このタイプのダッシュボードに関するユースケースの他の例には、インスタンス ID のドロップダウンを備えたメトリクスのダッシュボードを作成する変数として `InstanceId` を使用することが含まれます。あるいは、異なるリージョンの同じ一連のメトリクスを表示する変数として `region` を使用するダッシュボードを作成することもできます。

ダッシュボードプロパティ変数を使用して柔軟な Lambda ダッシュボードを作成するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで [ダッシュボード]、[Create dashboard] の順に選択します。
3. ダッシュボードの名前を入力し、[ダッシュボードを作成] を選択します。
4. Lambda 関数のメトリクスを表示するウィジェットをダッシュボードに追加します。これらのウィジェットを作成する際には、ウィジェットのメトリクスに [Lambda]、[関数名別] を指定します。関数には、このダッシュボードに含めるいずれかの Lambda 関数を指定します。

   ダッシュボードへのウィジェットの追加の詳細については、「CloudWatch ダッシュボードでウィジェットを作成して使用する」を参照してください。
5. ウィジェットを追加した後、ダッシュボードを表示しながら、[アクション]、[変数]、[変数を作成] の順に選択します。
6. [プロパティ変数] を選択します。
7. [変数が変更するプロパティ] で、[FunctionName] を選択します。
8. このユースケースでは、[入力タイプ] で [メニューを選択 (ドロップダウン)] を選択することをお勧めします。これにより、ダッシュボードにドロップダウンメニューが作成され、メトリクスを表示する Lambda 関数名を選択できます。

   これが、ある変数について、2 つまたは 3 つの異なる値を切り替えるダッシュボード用である場合は、[ラジオボタン] が適切な選択になります。

   変数の値を入力または貼り付けたい場合は、[テキスト入力] を選択します。このオプションにはドロップダウンリストやラジオボタンは含まれません。

9. [メニューを選択 (ドロップダウン)] を選択した場合は、メニューを設定するために、値を入力するか、またはメトリクス検索を使用するかを選択する必要があります。このユースケースでは、多数の Lambda 関数があり、それらのすべてを手動で入力することを望んでいないと仮定します。[メトリクス検索の結果を使用] を選択し、次の操作を実行します。

   a. [事前構築済みのクエリ]、[Lambda]、[エラー] を選択します。

      ([エラー] を選択しても、ダッシュボードには [エラー] メトリクスは追加されません。ただし、[FunctionName] 変数選択ボックスにはすぐに値が入力されます。)

   b. [関数名別] を選択し、[検索] を選択します。

      [検索] ボタンの下に、[FunctionName] が選択されていることが表示されます。また、入力ボックスにデータを入力するための値として見つかった、[FunctionName] ディメンションの値の数に関するメッセージも表示されます。

10. (オプション) その他の設定については、[二次設定] を選択し、次の 1 つ以上を実行します。

    - 変数の名前をカスタマイズするには、[カスタム変数名] に名前を入力します。
    - 変数入力フィールドのラベルをカスタマイズするには、[入力ラベル] にラベルを入力します。
    - ダッシュボードを最初に開いたときにこの変数のデフォルト値を設定するには、[デフォルト値] でデフォルト値を入力します。

11. [変数を追加] を選択します。

    [FunctionName] ドロップダウン選択ボックスがダッシュボードの上部近くに表示されます。このボックスで Lambda 関数を選択すると、その変数を使用するすべてのウィジェットで、選択した関数に関する情報が表示されます。

    後で、[FunctionName] ディメンションで Lambda メトリクスを監視するウィジェットをダッシュボードにさらに追加すると、それらのウィジェットは自動的にその変数を使用します。

チュートリアル: 正規表現パターンを使用してリージョンを切り替えるダッシュボードを作成する

この手順のステップでは、リージョンを切り替えることができる柔軟なダッシュボードを作成する方法を示します。このチュートリアルでは、プロパティ変数の代わりに正規表現のパターン変数を使用します。プロパティ変数を使用するチュートリアルについては、「チュートリアル: 関数名を変数として使用して Lambda ダッシュボードを作成する」を参照してください。

多くのユースケースでは、プロパティ変数を使用してリージョンを切り替えるダッシュボードを作成できます。ただし、ウィジェットがリージョン名を含む Amazon リソースネーム(ARN) に依存している場合は、パターン変数を使用して ARN 内のリージョン名を変更する必要があります。

ダッシュボードパターン変数を使用して柔軟なマルチリージョンダッシュボードを作成するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで [ダッシュボード]、[Create dashboard] の順に選択します。
3. ダッシュボードの名前を入力し、[ダッシュボードを作成] を選択します。
4. ダッシュボードにウィジェットを追加します。リージョン固有のデータを表示するウィジェットを追加する際には、1 つのリージョンでのみ表示される値を持つディメンションを指定しないでください。例えば、Amazon EC2 メトリクスの場合は、[InstanceID] をディメンションとして使用するメトリクスではなく、集計されたメトリクスを指定します。

   ダッシュボードへのウィジェットの追加の詳細については、「CloudWatch ダッシュボードでウィジェットを作成して使用する」を参照してください。
5. ウィジェットを追加した後、ダッシュボードを表示しながら、[アクション]、[変数]、[変数を作成] の順に選択します。
6. [パターン変数] を選択します。
7. [変数が変更するプロパティ] で、現在のダッシュボードリージョンの名前を入力します (例: **us-east-2**)。

   そのボックスの下のラベルに、変数の影響を受けるウィジェットが表示されていれば、正しいリージョンが入力されています。
8. このユースケースでは、[入力タイプ] で [ラジオボタン] を選択します。
9. [入力の設定方法を定義] で、[カスタム値のリストを作成] を選択します。
10. [カスタム値を作成] で、切り替えるリージョンを、1 行に 1 つずつ入力します。各リージョンの後にカンマを入力し、そのラジオボタン用に表示するラベルを入力します。例:

    ```
    us-east-1, N. Virginia

    us-east-2, Ohio

    eu-west-3, Paris
    ```

    カスタム値を入力すると、[プレビュー] ペインが更新され、ラジオボタンがどのように表示されるかが表示されます。

11. (オプション) その他の設定については、[二次設定] を選択し、次の 1 つ以上を実行します。

    - 変数の名前をカスタマイズするには、[カスタム変数名] に名前を入力します。
    - 変数入力フィールドのラベルをカスタマイズするには、[入力ラベル] にラベルを入力します。このチュートリアルでは、**`Region:`** と入力します。

      ここで値を入力すると、[プレビュー] ペインが更新され、ラジオボタンがどのように表示されるかが表示されます。
    - ダッシュボードを最初に開いたときにこの変数のデフォルト値を設定するには、[デフォルト値] でデフォルト値を入力します。
12. [変数を追加] を選択します。

    ダッシュボードが表示され、上部近くのリージョンのラジオボタンの横に [Region:] ラベルが表示されます。リージョンを切り替えると、該当の変数を使用するすべてのウィジェットで、選択したリージョンに関する情報が表示されます。

変数を別のダッシュボードにコピーする

便利な変数を含むダッシュボードを作成した後、これらの変数を他の既存のダッシュボードにコピーできます。ダッシュボード変数の詳細については、「ダッシュボード変数を使用して柔軟なダッシュボードを作成する」を参照してください。

ダッシュボード変数を別のダッシュボードにコピーするには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで、[ダッシュボード] を選択し、コピーする変数が含まれるダッシュボードの名前を選択します。必要に応じて、文字列を入力して、名前が一致するダッシュボードを検索します。
3. [アクション]、[変数]、[変数を管理] を選択します。
4. コピーする変数の横にあるラジオボタンを選択し、[別のダッシュボードにコピー] を選択します。
5. 選択ボックスを選択し、変数のコピー先とするダッシュボード名の入力を開始します。
6. ダッシュボード名を選択し、[変数をコピー] を選択します。

CloudWatch ダッシュボードでウィジェットを作成して使用する

このセクションのトピックを使用して、ダッシュボードでグラフ、アラーム、テキストウィジェットを作成して操作します。

目次

CloudWatch ダッシュボードからグラフを追加または削除する

CloudWatch ダッシュボードに、1 つ以上のメトリクスを含むグラフを追加できます。ダッシュボードに追加できるグラフのタイプは次のとおりです。[Line] (線)、[Stacked area] (スタックされたエリア)、[Number] (数値)、[Gauge] (ゲージ)、[Bar] (棒)、[Pie] (円)。不要になったグラフはダッシュボードから削除できます。このセクションでは、ダッシュボードでグラフを追加または削除する手順について説明します。ダッシュボードでグラフを編集する方法については、「CloudWatch ダッシュボードでグラフを編集する」を参照してください。

グラフをダッシュボードに追加する方法

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで [Dashboards] (ダッシュボード) を選択し、ダッシュボードをクリックします。

3. [+] 記号をクリックしてダッシュボードに追加するグラフのタイプを選択し、[次へ] をクリックします。

   - [Line] (線)、[Stacked area] (スタックされたエリア)、[Bar] (棒)、[Pie] (円) を選択した場合は、[Metrics] (メトリクス) を選択します。

4. [参照] タブで、グラフ化するメトリクスを検索または参照し、必要なメトリクスを選択します。
5. (オプション) グラフの時間範囲を変更するには、画面上部にある事前定義済みの時間範囲を選択します。時間範囲は、1 時間から 1 週間です ([1h] (1 時間)、[3h] (3時間)、[12h] (12 時間)、[1d] (1 日)、[3d] (3 日)、[1w] (1 週間))。

   独自の時間範囲を設定するには、[Custom] (カスタム) をクリックします。

   - (オプション) ダッシュボードの他の部分の時間範囲が後で変更されても、選択した時間範囲をウィジェットが使用し続けるようにするには、[持続時間範囲] を選択します。

6. (オプション) グラフのウィジェットタイプを変更するには、定義済みの時間範囲の横にあるドロップダウンを使用します。
7. (オプション) [Graphed metrics] (グラフ化したメトリクス) では、メトリクスに動的ラベルを追加して、メトリクスのラベル、ラベルの色、統計、期間を変更できます。Y 軸上のラベルの位置を、左から右に決定することもできます。

   a. 動的ラベルを追加するには、[Graphed metrics] (グラフ化したメトリクス)、[Add dynamic labels] (動的ラベルの追加) の順にクリックします。動的ラベルには、グラフの凡例のメトリクスに関する統計が表示されます。動的ラベルは、ダッシュボードまたはグラフが更新されるたびに自動的に更新されます。デフォルトでは、ラベルに追加した動的な値はラベルの先頭に表示されます。(詳細については、動的ラベルを使用する を参照してください)。
   b. メトリクスの色を変更するには、メトリクスの横にある色付きの四角を選択します。
   c. 統計を変更するには、[Statistic] (統計) の下にあるドロップダウンを選択し、新しい値を選択します。詳細については、「統計」を参照してください。
   d. 期間を変更するには、ドロップダウンで [Period] (期間) 列を選択してから、新しい値を選択します。

8. ゲージウィジェットを作成する場合は、[オプション] タブを選択し、ゲージの両端に使用する [最小] と [最大] の値を指定する必要があります。
9. (オプション) Y 軸をカスタマイズするには、[Option] (オプション) をクリックします。ラベルフィールドの [Left Y-axis] (左 Y 軸) にカスタムラベルを追加できます。グラフの Y 軸右側に値が表示されている場合は、そのラベルもカスタマイズできます。Y 軸の値に最小値と最大値を設定して、指定した値の範囲のみをグラフに表示することもできます。

10. (オプション) 折れ線グラフや積み上げ面グラフに水平注釈を追加または編集したり、ゲージウィジェットにしきい値を追加したりするには、[オプション] を選択します。

    a. 水平注釈またはしきい値を追加するには、[水平注釈の追加] または [しきい値を追加] を選択します。

    b. [ラベル] に注釈のラベルを入力し、チェックマークアイコンをクリックします。

    c. [Value] (値) で、現在の値の横にあるペンと紙のアイコンをクリックし、新しい値を入力します。値を入力したら、チェックマークアイコンをクリックします。

    d. [Fill] (塗りつぶし) で、ドロップダウンを選択し、注釈でシェーディングを使用する方法を指定します。[None] (なし)、[Above] (上回る)、[Between] (間)、[Below] (下回る) を選択できます。塗りつぶしの色を変更するには、注釈の横の色付きの四角を選択します。

    e. [Axis] (軸) で、注釈を Y 軸の左側と右側のどちらに表示するかを指定します。

    f. 注釈を非表示にするには、非表示にする注釈の横にあるチェックボックスをオフにします。

    g. 注釈を削除するには、[Actions] の [X] を選択します。

    > **Note**
    >
    > これらのステップを繰り返すと、同じグラフやゲージに複数の水平注釈やしきい値を追加することができます。

11. (オプション) 垂直の注釈を追加または編集するには、[Option] (オプション) をクリックします。

    a. 垂直の注釈を追加するには、[垂直の注釈の追加] を選択します。

    b. [Label] (ラベル) で、現在の注釈の横にあるペンと紙のアイコンをクリックし、新しい注釈を入力します。日付と時刻のみを表示するには、[Label] (ラベル) フィールドを空白のままにします。

    c. [Date] (日付) で、現在の日付と時刻を選択し、新しい日付と時刻を入力します。

    d. [Fill] (塗りつぶし) で、ドロップダウンを選択し、注釈でシェーディングを使用する方法を指定します。[None] (なし)、[Above] (上回る)、[Between] (間)、[Below] (下回る) を選択できます。塗りつぶしの色を変更するには、注釈の横にある色の四角を選択します。

    e. 注釈を非表示にするには、非表示にする注釈の横にあるチェックボックスをオフにします。

    f. 注釈を削除するには、[Actions] の [X] を選択します。

> **Note**
>
> これらのステップを繰り返すと、同じグラフに複数の垂直注釈を追加することができます。

12. [ウィジェットの作成] を選択します。
13. [ダッシュボードの保存] を選択します。

グラフをダッシュボードから削除する方法

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで [Dashboards] (ダッシュボード) を選択し、ダッシュボードをクリックします。
3. 削除するグラフの右上隅の [Widget actions] (ウィジェットのアクション) を選択してから、[Delete] (削除) をクリックします。
4. [ダッシュボードの保存] を選択します。

CloudWatch ダッシュボードで手動でメトリクスをグラフ化する

過去 14 日間にわたってデータを発行していないメトリクスは、グラフに追加するメトリクスを CloudWatch ダッシュボードで検索したときに、検出されません。既存のグラフにメトリクスを手動で追加するには、次のステップを使用します。

検索で見つからないメトリクスをグラフに追加するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで [Dashboards] (ダッシュボード) を選択し、ダッシュボードをクリックします。
3. ダッシュボードには、メトリクスを追加する先のグラフが既に含まれている必要があります。含まれていない場合は、グラフを作成して、メトリクスを追加します。詳細については、「CloudWatch ダッシュボードからグラフを追加または削除する」を参照してください。
4. [アクション]、[ソースの表示/編集] を選択します。

JSON ブロックが表示されます。このブロックは、ダッシュボード上のウィジェットとそのコンテンツを指定します。このブロックの一部の例を次に示します。これは 1 つのグラフを定義します。

```
{
            "type": "metric",
            "x": 0,
            "y": 0,
            "width": 6,
            "height": 3,
            "properties": {
                "view": "singleValue",
                "metrics": [
                    [ "AWS/EBS", "VolumeReadOps", "VolumeId",
 "vol-1234567890abcdef0" ]
                ],
                "region": "us-west-1"
            }
        },
```

この例では、以下のセクションで、このグラフに表示されるメトリクスを定義します。

```
[ "AWS/EBS", "VolumeReadOps", "VolumeId", "vol-1234567890abcdef0" ]
```

5. 最後の括弧の後にカンマを追加し (まだ追加されていない場合)、カンマの後に同じような括弧で囲んだセクションを追加します。この新しいセクションでは、グラフに追加するメトリクスの名前空間、メトリクス名、および必要なディメンションを指定します。次に例を示します。

```
[ "AWS/EBS", "VolumeReadOps", "VolumeId", "vol-1234567890abcdef0" ],
[ "MyNamespace", "MyMetricName", "DimensionName", "DimensionValue" ]
```

JSON でメトリクスをフォーマットする方法の詳細については、「メトリクスウィジェットオブジェクトのプロパティ」を参照してください。

6. [Update] (更新) を選択します。

既存のグラフの使用

これらのセクションの手順に従って、既存のダッシュボードグラフウィジェットを編集および変更します。

トピック

CloudWatch ダッシュボードでグラフを編集する

CloudWatch ダッシュボードに追加するグラフを編集できます。グラフのタイトル、統計、期間を変更できます。グラフでメトリクスを追加、更新、削除できます。グラフに 1 つ以上のメトリクスが含まれる場合は、使用していないメトリクスを非表示にして、整理することができます。このセクションの手順では、ダッシュボードのグラフを編集する方法について説明します。グラフの作成方法については、「CloudWatch ダッシュボードからグラフを追加または削除する」を参照してください。

New interface

ダッシュボードのグラフを編集する方法

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで [Dashboards] (ダッシュボード) を選択し、ダッシュボードをクリックします。
3. 編集するグラフの右上隅の [Widget actions] (ウィジェットのアクション) を選択してから、[Edit] (編集) をクリックします。
4. グラフのタイトルを変更するには、現在のタイトルの横にあるペンと紙のアイコンをクリックします。新しいタイトルを入力して、[Apply] (適用) を選択します。
5. (オプション) グラフの時間範囲を変更するには、グラフの上部にある事前定義済みの時間範囲を選択します。時間範囲は、1 時間から 1 週間です ([1h] (1 時間)、[3h] (3時間)、[12h] (12 時間)、[1d] (1 日)、[3d] (3 日)、[1w] (1 週間))。

   独自の時間範囲を設定するには、[Custom] (カスタム) をクリックします。

- (オプション) ダッシュボードの他の部分の時間範囲が後で変更されても、選択した時間範囲をウィジェットが使用し続けるようにするには、[持続時間範囲] を選択します。

6. グラフのウィジェットタイプを変更するには、定義済みの時間範囲の横にあるドロップダウンを使用します。
7. [Graphed metrics] (グラフ化したメトリクス) では、メトリクスに動的ラベルを追加し、メトリクスのラベル、ラベルの色、統計、期間を変更できます。Y 軸上のラベルの位置を、左から右に決定することもできます。

   a. メトリクスに動的ラベルを追加するには、[Dynamic labels] (動的ラベル) を選択します。動的ラベルは、グラフの凡例にメトリクスに関する統計を表示します。動的ラベルは、ダッシュボードまたはグラフが更新されるたびに自動的に更新されます。デフォルトでは、ラベルに追加した動的な値は、ラベルの先頭に表示されます。詳細については、「動的ラベルを使用する」を参照してください。

   b. メトリクスの色を変更するには、メトリクスの横にある色付きの四角を選択します。

   c. 統計を変更するには、[Statistic] (統計) 列で統計値を選択してから、新しい値を選択します。詳細については、「統計」を参照してください。

   d. 期間を変更するには、[Period] (期間) 列で期間の値を選択してから、新しい値を選択します。

8. 水平の注釈を追加または編集するには、[Options] (オプション) を選択します。

   a. 水平の注釈を追加するには、[水平の注釈の追加] を選択します。

   b. [Label] (ラベル) で、現在の注釈の横にあるペンと紙のアイコンをクリックします。次に、新しい注釈を入力します。注釈を入力したら、チェックマークアイコンをクリックします。

   c. [Value] (値) で、現在のメトリクス値の横にあるペンと紙のアイコンをクリックします。次に、新しいメトリクス値を入力します。値を入力したら、チェックマークアイコンをクリックします。

   d. [Fill] (フィル) で、列の下のドロップダウンを選択し、注釈でシェーディングを使用する方法を指定します。[None] (なし)、[Above] (上回る)、[Between] (間)、[Below] (下回る) を選択できます。[Between] (間) を選択すると、別の新しいラベルと値フィールドが表示されます。

> **Tip**
>
> 注釈の横にある色付きの四角を選択すると、塗りつぶしの色を変更できます。

e. [Axis] (軸) で、注釈を Y 軸の左側と右側のどちらに表示するかを指定します。

f. 注釈を非表示にするには、グラフで非表示にする注釈の横にあるチェックボックスをオフにします。

g. 注釈を削除するには、[Actions] (アクション) 列の [X] を選択します。

> **Note**
>
> これらのステップを繰り返すと、同じグラフに複数の水平注釈を追加することができます。

9. 垂直の注釈を追加または編集するには、[Options] (オプション) を選択します。

a. 垂直の注釈を追加するには、[垂直の注釈の追加] を選択します。

b. [Label] (ラベル) で、現在の注釈の横にあるペンと紙のアイコンをクリックします。次に、新しい注釈を入力します。注釈を入力したら、チェックマークアイコンをクリックします。

> **Tip**
>
> 注釈で日付と時刻のみを表示するには、[Label] (ラベル) フィールドを空白のままにします。

c. [Date] (日付) で、現在の日付と時刻を選択します。次に、新しい日付と時刻を入力します。

d. [Fill] (フィル) で、列の下のドロップダウンを選択し、注釈でシェーディングを使用する方法を指定します。[None] (なし)、[Above] (上回る)、[Between] (間)、[Below] (下回る) を選択できます。[Between] (間) を選択すると、新しいラベルと値フィールドが表示されます。

> **Tip**
>
> 注釈の横の色付きの四角を選択すると、塗りつぶしの色を変更できます。

> **Note**
>
> これらのステップを繰り返すと、同じグラフに複数の垂直注釈を追加することができます。

e. 注釈を非表示にするには、グラフで非表示にする注釈の横にあるチェックボックスをオフにします。

f. 注釈を削除するには、[Actions] (アクション) 列の [X] を選択します。

10. Y 軸をカスタマイズするには、[Option] (オプション) を選択します。[Left Y-axis] (左 Y 軸) の [Label] (ラベル) にカスタムラベルを入力できます。グラフに右 Y 軸の値が表示されている場合は、そのラベルもカスタマイズできます。Y 軸の値に最小値と最大値を設定して、指定した値の範囲のみをグラフに表示することもできます。

11. 変更が完了したら、[Update widget] (ウィジェットの更新) を選択します。

グラフの凡例を非表示にしたり位置を変更したりするには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで [Dashboards] (ダッシュボード) を選択し、ダッシュボードをクリックします。
3. 編集するグラフの右上隅の [Widget actions] (ウィジェットのアクション) を選択します。[Legend] (凡例) を選択し、[Hidden] (非表示)、[Bottom] (下へ)、[Right] (右) から選択します。

一時的にダッシュボードのグラフメトリクスを非表示にすることができます。

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで [Dashboards] (ダッシュボード) を選択し、ダッシュボードをクリックします。
3. グラフのフッターで非表示にするメトリクスの色の四角を選択します。カーソルを合わせると、色付きの四角に [X] が表示され、クリックするとグレーに変わります。
4. 非表示のメトリクスを復元するには、グレーの四角の中の [X] をクリアします。

Original interface

ダッシュボードのグラフを編集する方法

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで [Dashboards] (ダッシュボード) を選択し、ダッシュボードをクリックします。
3. 編集するグラフの右上隅にカーソルを合わせます。[Widget Actions] (ウィジェットのアクション) を選択してから、[Edit] (編集) を選択します。
4. グラフのタイトルを変更するには、現在のタイトルの横にある鉛筆アイコンをクリックし、新しいタイトルを入力します。
5. グラフの時間範囲を変更するには、グラフの上部にある事前定義済みの時間範囲の 1 つを選択します。これらの期間は、1 時間から 1 週間です ([1h] (1 時間)、[3h] (3 時間)、[12h] (12 時間)、[1d] (1 日)、[3d] (3 日)、[1w] (1 週間))。

   - 独自の時間範囲を設定するには、[custom] (カスタム) をクリックします。
6. グラフのウィジェットタイプを変更するには、[Graph options] (グラフのオプション) タブを選択します。[Line] (折れ線グラフ)、[Stacked area] (スタックされたエリア)、[Number] (数値)、[Bar] (棒グラフ)、[Pie] (円グラフ) を選択できます。

   > **Tip**
   >
   > 定義済みの時間範囲の横にあるドロップダウンを選択して、グラフのウィジェットタイプを変更できます。

7. [Graphed metrics] (グラフ化したメトリクス) では、メトリクスに動的ラベルを追加し、メトリクスのラベル、ラベルの色、統計、期間を変更できます。Y 軸上のラベルの位置を、左から右に決定することもできます。

   a. メトリクスに動的ラベルを追加するには、[Dynamic labels] (動的ラベル) を選択します。動的ラベルは、グラフの凡例にメトリクスに関する統計を表示します。動的ラベルは、ダッシュボードまたはグラフが更新されるたびに自動的に更新されます。デフォルトでは、ラベルに追加した動的な値は、ラベルの先頭に表示されます。詳細については、「動的ラベルを使用する」を参照してください。

   b. メトリクスの色を変更するには、メトリクスの横にある色付きの四角を選択します。

   c. 統計を変更するには、[Statistic] (統計) 列で統計値を選択してから、新しい値を選択します。詳細については、「統計」を参照してください。

d. 期間を変更するには、[Period] (期間) 列で期間の値を選択してから、新しい値を選択します。

8. 水平の注釈を追加または編集するには、[グラフのオプション] を選択します。

a. 水平の注釈を追加するには、[水平の注釈の追加] を選択します。

b. [Label] (ラベル) で、現在の注釈の横にある鉛筆アイコンをクリックします。次に、新しい注釈を入力します。注釈を入力したら、チェックマークアイコンをクリックします。

c. [Value] (値) で、現在のメトリクス値の横にある鉛筆アイコンをクリックします。次に、新しいメトリクス値を入力します。値を入力したら、チェックマークアイコンをクリックします。

d. [Fill] (フィル) で、列の下のドロップダウンを選択し、注釈でシェーディングを使用する方法を指定します。[None] (なし)、[Above] (上回る)、[Between] (間)、[Below] (下回る) を選択できます。[Between] (間) を選択すると、新しいラベルと値フィールドが表示されます。

> **Tip**
>
> 注釈の横の色付きの四角を選択すると、塗りつぶしの色を変更できます。

e. [Axis] (軸) で、注釈を Y 軸の左側と右側のどちらに表示するかを指定します。

f. 注釈を非表示にするには、グラフで非表示にする注釈の横にあるチェックボックスをオフにします。

g. 注釈を削除するには、[Actions] (アクション) 列の [X] を選択します。

> **Note**
>
> これらのステップを繰り返すと、同じグラフに複数の水平注釈を追加することができます。

9. 水平注釈を追加または編集するには、[Graph options] (グラフのオプション) を選択します。

a. 垂直の注釈を追加するには、[垂直の注釈の追加] を選択します。

b. [Label] (ラベル) で、現在の注釈の横にある鉛筆アイコンをクリックします。次に、新しい注釈を入力します。注釈を入力したら、チェックマークアイコンをクリックします。

> **Tip**
>
> 注釈で日付と時刻のみを表示するには、[Label] (ラベル) フィールドを空白のままにします。

c. [Date] (日付) で、現在の日付と時刻の横にある鉛筆アイコンをクリックします。次に、新しい日付と時刻を入力します。

d. [Fill] (フィル) で、列の下のドロップダウンを選択し、注釈でシェーディングを使用する方法を指定します。[None] (なし)、[Above] (上回る)、[Between] (間)、[Below] (下回る) を選択できます。[Between] (間) を選択すると、新しいラベルと値フィールドが表示されます。

> **Tip**
>
> 注釈の横の色付きの四角を選択すると、塗りつぶしの色を変更できます。

> **Note**
>
> これらのステップを繰り返すと、同じグラフに複数の垂直注釈を追加することができます。

e. 注釈を非表示にするには、グラフで非表示にする注釈の横にあるチェックボックスをオフにします。

f. 注釈を削除するには、[Actions] (アクション) 列の [X] を選択します。

10. Y 軸をカスタマイズするには、[グラフのオプション] を選択します。[Left Y-axis] (左 Y 軸) の [Label] (ラベル) にカスタムラベルを入力できます。グラフに右 Y 軸の値が表示されている場合は、そのラベルもカスタマイズできます。Y 軸の値に最小値と最大値を設定して、指定した値の範囲のみをグラフに表示することもできます。

11. 変更が完了したら、[Update widget] (ウィジェットの更新) を選択します。

グラフの凡例を非表示にしたり位置を変更したりするには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。

2. ナビゲーションペインで [Dashboards] (ダッシュボード) を選択し、ダッシュボードをクリックします。
3. 編集するグラフの右上隅にカーソルを合わせて、[Widget actions] (ウィジェットのアクション) をクリックします。[Legend] (凡例) を選択し、[Hidden] (非表示)、[Bottom] (下へ)、[Right] (右) から選択します。

一時的にダッシュボードのグラフメトリクスを非表示にすることができます。

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで [Dashboards] (ダッシュボード) を選択し、ダッシュボードをクリックします。
3. グラフのフッターで非表示にするメトリクスの色の四角を選択します。カーソルを合わせると、色付きの四角に [X] が表示され、クリックするとグレーに変わります。
4. 非表示のメトリクスを復元するには、グレーの四角の中の [X] をクリアします。

CloudWatch ダッシュボードでグラフを移動またはサイズ変更する

CloudWatch ダッシュボードのグラフを調節したり、サイズを変更したりすることができます。

ダッシュボードのグラフを移動する方法

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで [Dashboards] (ダッシュボード) を選択し、ダッシュボードをクリックします。
3. 次のいずれかを行います。
   - 選択アイコンが表示されるまで、グラフのタイトルにマウスカーソルを合わせます。グラフを選択し、ダッシュボード上の移動先にドロップします。
   - ウィジェットをダッシュボードの左上または左下に移動するには、ウィジェットの右上にある縦の省略記号を選択して、[ウィジェットのアクション] メニューを開きます。次に [移動] を選択し、ウィジェットの移動先を選択します。
4. [ダッシュボードの保存] を選択します。

グラフのサイズを変更するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。

2. ナビゲーションペインで [Dashboards] (ダッシュボード) を選択し、ダッシュボードをクリックします。
3. サイズを大きくしたり小さくするには、グラフにマウスカーソルを合わせ、グラフの右下をドラッグします。目的のサイズになったら、コーナーのドラッグを停止します。
4. [ダッシュボードの保存] を選択します。

一時的にグラフを拡大するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで [Dashboards] (ダッシュボード) を選択し、ダッシュボードをクリックします。
3. グラフを選択します。あるいは、グラフのタイトルにマウスカーソルを合わせ、[ウィジェットアクション] を選択し、[拡大] を選択します。

CloudWatch ダッシュボードのグラフ名を変更する

CloudWatch がダッシュボードのグラフに対してつけたデフォルト名を変更できます。

ダッシュボードのグラフ名を変更する方法

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで [Dashboards] (ダッシュボード) を選択し、ダッシュボードをクリックします。
3. グラフのタイトルにマウスカーソルを合わせ、[ウィジェットのアクション]、[編集] の順に選択します。
4. [グラフの編集] 画面の上部で、グラフのタイトルを選択します。
5. [タイトル] に新しい名前を入力し、[OK] (チェックマーク) を選択します。[グラフの編集] 画面の右下にある [ウィジェットの更新] を選択します。

CloudWatch ダッシュボードへのメトリクスエクスプローラーウィジェットを追加する

メトリクスエクスプローラーウィジェットには、同じタグを持つ複数のリソースのグラフや、インスタンスタイプなどの同じリソースプロパティを共有する複数のリソースのグラフが含まれます。一致するリソースの作成または削除に伴って、これらのウィジェットも最新の状態になります。ダッシュ

ボードにメトリクスエクスプローラーウィジェットを追加すると、環境のトラブルシューティングをより効率的に行うのに役立ちます。

例えば、本番環境やテストなどの環境を表すタグを割り当てることで、EC2 インスタンスのフリートをモニターリングできます。その後、これらのタグを使用して、各タグに関連付けられている EC2 インスタンスのヘルスやパフォーマンスを把握するための `CPUUtilization` などの運用メトリクスをフィルタリングおよび集計できます。

次の手順では、コンソールを使用してメトリクスエクスプローラーウィジェットをダッシュボードに追加する方法について説明します。プログラムまたは を使用して追加することもできますAWS CloudFormation 詳細については、「メトリクスエクスプローラーウィジェットのオブジェクト定義」および「AWS::CloudWatch::Dashboard」を参照してください。

ダッシュボードにメトリクスエクスプローラーウィジェットを追加するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで、ダッシュボードを選択します。
3. メトリクスエクスプローラーウィジェットを追加するダッシュボードの名前を選択します。
4. [+] 記号を選択します。
5. [Explorer] (エクスプローラー) を選択し、[Next] (次へ) を選択します。

   > **Note**
   >
   > メトリクスエクスプローラーウィジェットを追加できるようにするには、新しいダッシュボードビューを選択している必要があります。オプトインするには、ナビゲーションペインで [ダッシュボード] を選択し、ページの上部にあるバナーで [新しいインターフェイスを試す] を選択します。

6. 次のいずれかを行ってください。
   - テンプレートを使用するには、[Pre-filled Explorer widget] (事前入力済みのエクスプローラーウィジェット) を選択し、使用するテンプレートを選択します。
   - カスタムビジュアライゼーションを作成するには、[Empty Explorer widget] (空のエクスプローラーウィジェット) を選択します。
7. [作成] を選択します。

テンプレートを使用した場合、ウィジェットは選択したメトリクスとともにダッシュボードに表示されます。エクスプローラーウィジェットとダッシュボードに問題がなければ、[Save dashboard] (ダッシュボードの保存) を選択します。

テンプレートを使用しなかった場合は、次の手順に進みます。

8. [Explorer] (エクスプローラー) の下の新しいウィジェットの [Metrics] (メトリクス) ボックスで、サービスから 1 つのメトリクスまたは使用可能なすべてのメトリクスを選択します。

   メトリクスを選択した後、必要に応じてこのステップを繰り返して、さらにメトリクスを追加できます。

9. 選択した各メトリクスについて、CloudWatch は、メトリクス名の直後に使用する統計を表示します。これを変更するには、統計名を選択し、必要な統計を選択します。

10. [From] (開始) で、結果をフィルタリングするためのタグまたはリソースプロパティを選択します。

    これを実行した後、必要に応じてこのステップを繰り返して、さらにタグまたはリソースプロパティを選択できます。

    2 つの EC2 インスタンスタイプなど、同じプロパティの複数の値を選択した場合、Explorer には、選択したいずれかのプロパティに一致するすべてのリソースが表示されます。OR 演算として取り扱われます。

    `Production` タグや M5 インスタンスタイプなど、異なるプロパティまたはタグを選択した場合、これらのすべての選択に一致するリソースのみが表示されます。これは、AND 演算として扱われます。

11. (オプション) [Aggregate by] (集計単位) で、メトリクスの集計に使用する統計を選択します。その後、[for] (次のため:) の横で、リストからメトリクスの集計方法を選択します。現在表示されているすべてのリソースを集計することも、単一のタグまたはリソースプロパティ別に集計することもできます。

    集計方法に応じて、結果は 1 つの時系列または複数の時系列になります。

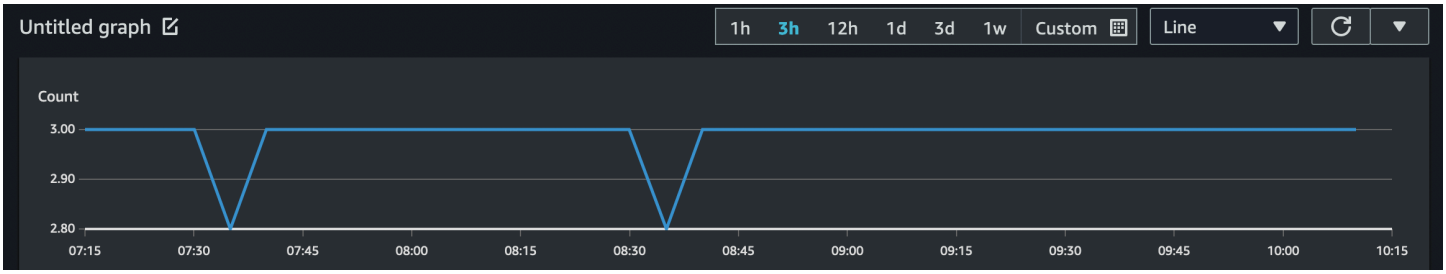
12. [Split by] (分割方法) では、複数の時系列を持つ単一のグラフを複数のグラフに分割できます。分割は、[Split by] (分割方法) で選択するさまざまな基準によって行うことができます。

13. [Graph options] (グラフのオプション) で、期間、グラフのタイプ、凡例の配置、およびレイアウトを変更することにより、より洗練されたグラフを表示できます。

14. エクスプローラーウィジェットとダッシュボードに問題がなければ、[Save dashboard] (ダッシュボードの保存) を選択します。

CloudWatch ダッシュボードで線ウィジェットを追加または削除する

線ウィジェットを使用すると、一定期間にわたってメトリクスを比較できます。ウィジェットのミニマップズーム機能を使用して、ズームインビューとズームアウトビューを変更することなく、折れ線グラフのセクションを調べることもできます。このセクションの手順では、CloudWatch ダッシュボードで線ウィジェットを追加および削除する方法について説明します。ウィジェットのミニマップズーム機能を折れ線グラフで使用する方法については、「Zooming in on a line or stacked area graph」(折れ線グラフまたは積み上げ面グラフでのズームイン) を参照してください。



線ウィジェットをダッシュボードに追加するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで [Dashboards] (ダッシュボード) を選択し、ダッシュボードをクリックします。
3. [+] 記号を選択してから、[Line] を選択します。
4. [メトリクス] をクリックします。
5. [Browse] (参照) をクリックし、グラフ化するメトリクスを選択します。
6. [Create widget] (ウィジェットの作成)、[Save dashboard] (ダッシュボードの保存) の順にクリックします。

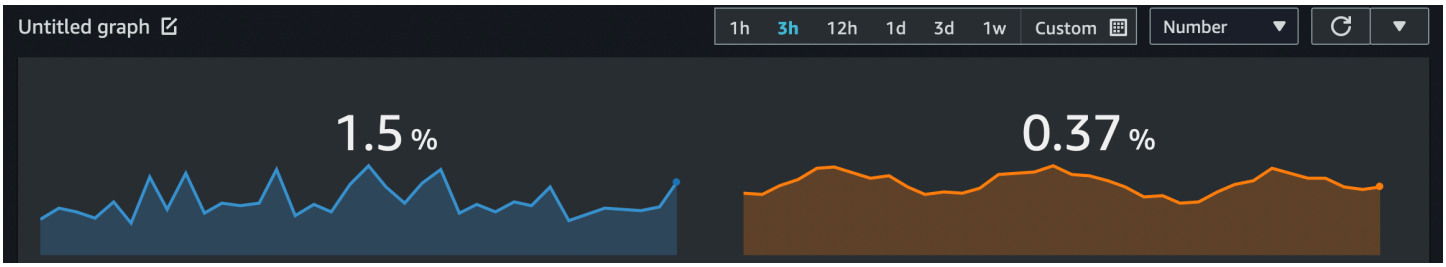
線ウィジェットをダッシュボードから削除するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで [Dashboards] (ダッシュボード) を選択し、ダッシュボードをクリックします。

3. 削除する線ウィジェットの右上隅で、[Widget actions] (ウィジェットのアクション)、[Delete] (削除) の順にクリックします。
4. [ダッシュボードの保存] を選択します。

CloudWatch ダッシュボードから数値ウィジェットを追加または削除する

数値ウィジェットを使用すると、表示される最新のメトリクス値とトレンドをすぐに確認できます。数値ウィジェットにはスパークライン機能が含まれているため、メトリクストレンドの上半分と下半分を 1 つのグラフで可視化できます。このセクションの手順では、CloudWatch ダッシュボードで数値ウィジェットを追加または削除する方法について説明します。



> **Note**
>
> スパークライン機能をサポートしているのは、新しいインターフェイスのみです。数値ウィジェットを作成すると、スパークライン機能が自動的に含まれます。

数値ウィジェットをダッシュボードに追加するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで [Dashboards] (ダッシュボード) を選択し、ダッシュボードをクリックします。
3. [+] 記号を選択してから、[Number] を選択します。
4. [参照] タブで、表示するメトリクスを検索または参照します。
5. (オプション) スパークライン機能の色を変更するには、[Graphed metrics] (グラフ化したメトリクス) をクリックし、メトリクスラベルの横にあるカラーボックスを選択します。別の色を選択するか、6 桁の 16 進数カラーコードを入力して色を指定できるメニューが表示されます。
6. (オプション) スパークライン機能をオフにするには、[Option] (オプション) をクリックします。[Sparkline] (スパークライン) のチェックボックス。

7. (オプション) 数値ウィジェットの時間範囲を変更するには、ウィジェットの上部にある事前定義済みの時間範囲を選択します。時間範囲は、1 時間から 1 週間です ([1h] (1 時間)、[3h] (3時間)、[12h] (12 時間)、[1d] (1 日)、[3d] (3 日)、[1w] (1 週間))。

   独自の時間範囲を設定するには、[Custom] (カスタム) をクリックします。

   - (オプション) ダッシュボードの他の部分の時間範囲が後で変更されても、選択した時間範囲をウィジェットが使用し続けるようにするには、[持続時間範囲] を選択します。

8. (オプション) 数値ウィジェットに集計を表示するには ([1 時間]、[3 時間]、[12 時間]、[1 日]、[3 日]、[1 週間])。

   独自の時間範囲を設定するには、[Custom] (カスタム) をクリックします。

   - (オプション) このウィジェットに最新の値ではなく、時間範囲全体のメトリクス値の平均を表示させるには、[オプション] で [時間範囲の値は時間範囲全体の値を表示する] を選択します。

9. [Create widget] (ウィジェットの作成)、[Save dashboard] (ダッシュボードの保存) の順にクリックします。

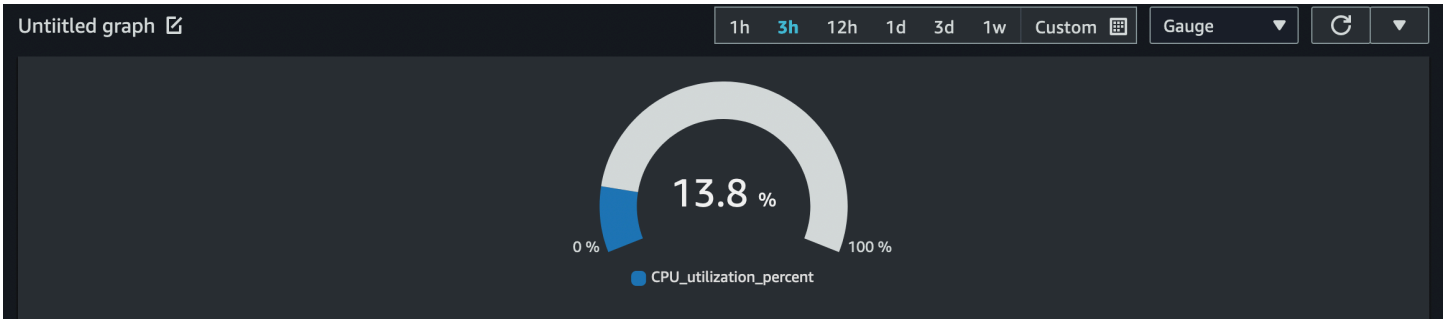
> **Tip**
>
> ダッシュボード画面の数値ウィジェットで、スパークライン機能をオフにすることができます。変更する数値ウィジェットの右上隅で、[Widget actions] (ウィジェットのアクション) をクリックします。[Sparkline] (スパークライン)、[Hide sparkline] (スパークラインを非表示) の順に選択します。

数値ウィジェットをダッシュボードから削除するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで [Dashboards] (ダッシュボード) をクリックし、削除する数値ウィジェットを含むダッシュボードを選択します。
3. 削除する数値ウィジェット右上隅で、[Widget actions] (ウィジェットのアクション)、[Delete] (削除) の順にクリックします。
4. [ダッシュボードの保存] を選択します。

CloudWatch ダッシュボードでゲージウィジェットを追加または削除する

ゲージウィジェットを使用すると、範囲間のメトリクス値を可視化できます。たとえば、ゲージウィジェットを使用してパーセンテージと CPU 使用率をグラフ化し、発生するパフォーマンスの問題を観察して診断できます。このセクションの手順では、CloudWatch ダッシュボードでゲージウィジェットを追加または削除する方法について説明します。



> **Note**
>
> CloudWatch コンソールの新しいインターフェイスのみが、ゲージウィジェットの作成をサポートしています。このウィジェットを作成する際は、ゲージ範囲を設定する必要があります。

ゲージウィジェットをダッシュボードに追加するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで [Dashboards] (ダッシュボード) を選択し、ダッシュボードをクリックします。
3. ダッシュボード画面で、+ 記号を選択してから、[Gauge] (ゲージ) を選択します。
4. [Browse] (参照) をクリックし、グラフ化するメトリクスを選択します。
5. [Options] を選択します。[Gauge range] (ゲージ範囲)で、[Min] (最小) および [Max] (最大) 値を設定します。CPU 使用率などのパーセンテージについては、`Min` に `0`、`Max` に `100` の値を設定することをお勧めします。
6. (オプション) ゲージウィジェットの色を変更するには、[Graphed metrics] (グラフ化したメトリクス) をクリックし、メトリクスラベルの横にあるカラーボックスを選択します。別の色を選択するか、6 桁の 16 進数カラーコードを入力して色を指定できるメニューが表示されます。
7. [Create widget] (ウィジェットの作成)、[Save dashboard] (ダッシュボードの保存) の順にクリックします。

ダッシュボードからゲージウィジェットを削除するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで [Dashboards] (ダッシュボード) をクリックし、削除するゲージウィジェットを含むダッシュボードを選択します。
3. 削除するゲージウィジェットの右上隅で、[Widget actions] (ウィジェットのアクション)、[Delete] (削除) の順にクリックします。
4. [ダッシュボードの保存] を選択します。

CloudWatch ダッシュボードにカスタムウィジェットを追加する

カスタムウィジェットは、カスタムパラメータを使用して任意の AWS Lambda 関数を呼び出すことができる CloudWatch ダッシュボードウィジェットです。次に、返された HTML または JSON を表示します。カスタムウィジェットは、ダッシュボードにデータのカスタムビューを構築する簡単な方法です。Lambda コードを書けて HTML を作成できれば、便利なカスタムウィジェットを作成できます。さらに、Amazon では、コードなしで作成できる、ビルド済みのカスタムウィジェットをいくつかご用意しています。

カスタムウィジェットとして使用する Lambda 関数を作成するときは、関数名にプレフィックス customWidget を含めることを強くお勧めします。これにより、ダッシュボードにカスタムウィジェットを追加するときに安全に使用できる Lambda 関数がわかります。

カスタムウィジェットは、ダッシュボード上の他のウィジェットと同様に動作します。これらのオブジェクトは、再表示、自動更新、サイズ変更、および移動することができます。これらは、ダッシュボードの時間範囲に応答します。

CloudWatch コンソールのクロスアカウント機能を設定している場合は、1 つのアカウントで作成されたカスタムウィジェットを他のアカウントのダッシュボードに追加できます。詳細については、「クロスアカウントクロスリージョン CloudWatch コンソール」を参照してください。

CloudWatch ダッシュボードの共有機能を使用して、独自のウェブサイトでカスタムウィジェットを使用することもできます。詳細については、「CloudWatch ダッシュボードの共有」を参照してください。

トピック

- カスタムウィジェットの詳細
- セキュリティと JavaScript
- カスタムウィジェットのインタラクティブ性

- カスタムウィジェットを作成する
- カスタムウィジェットのサンプル

カスタムウィジェットの詳細

カスタムウィジェットでは次のようなことができます。

1. CloudWatch ダッシュボードは、ウィジェットコードを含む Lambda 関数を呼び出します。これは、ウィジェットで定義された任意のカスタムパラメータを渡します。
2. Lambda 関数は、HTML、JSON、または Markdown の文字列を返します。Markdown は、次の形式で JSON として返されます。

```
{"markdown":"markdown content"}
```

3. ダッシュボードに、返された HTML または JSON が表示されます。

関数が HTML を返す場合、ほとんどの HTML タグがサポートされます。カスケードスタイルシート (CSS) スタイルとスケーラブルベクターグラフィックス (SVG) を使用すれば、洗練されたビューを構築できます。

リンクやテーブルなどの HTML 要素のデフォルトのスタイルは、CloudWatch ダッシュボードのスタイルに従います。<style> タグを使用してインラインスタイルを使用することで、このスタイルをカスタマイズできます。また、cwdb-no-default-styles のクラスで単一の HTML 要素を含めることで、デフォルトのスタイルを無効化することができます。次の の例では、デフォルトのスタイルを無効化します。

カスタムウィジェットから Lambda へのすべての呼び出しには、以下の内容の widgetContext 要素が含まれ、Lambda 関数のデベロッパーに有用なコンテキスト情報を提供します。

```
{
    "widgetContext": {
        "dashboardName": "Name-of-current-dashboard",
        "widgetId": "widget-16",
        "accountId": "012345678901",
        "locale": "en",
        "timezone": {
            "label": "UTC",
            "offsetISO": "+00:00",
            "offsetInMinutes": 0
```

```
        },
        "period": 300,
        "isAutoPeriod": true,
        "timeRange": {
            "mode": "relative",
            "start": 1627236199729,
            "end": 1627322599729,
            "relativeStart": 86400012,
            "zoom": {
                "start": 1627276030434,
                "end": 1627282956521
             }
        },
        "theme": "light",
        "linkCharts": true,
        "title": "Tweets for Amazon website problem",
        "forms": {
            "all": {}
        },
        "params": {
            "original": "param-to-widget"
        },
        "width": 588,
        "height": 369
    }
}
```

デフォルトの CSS スタイル設定

カスタムウィジェットには、次のデフォルトの CSS スタイルの要素が用意されています。

- CSS クラスの btn を使用してボタンを追加します。これは、次の例のように、アンカー (<a>) をボタンに変えます。

```
<a class="btn" href=https://amazon.com”>Open Amazon</a>
```

- CSS クラスの btn btn-primary を使用して、主ボタンを追加できます。
- デフォルトでは、テーブル、選択、ヘッダー (h1、h2、h3)、書式付きテキスト (前)、インプットおよびテキスト領域の要素にスタイルが設定されます。

describe パラメータを使用する

空の文字列を返すだけの場合でも、関数の describe パラメータをサポートすることを強くお勧めします。サポートされていない状態でカスタムウィジェットで呼び出されると、ウィジェットの内容がドキュメントのような形式で表示されます。

describe パラメータを含めると、Lambda 関数はドキュメントを Markdown 形式で返し、他には何も実行しません。

コンソールでカスタムウィジェットを作成する場合、Lambda 関数を選択すると、[Get documentation] (ドキュメントを取得) ボタンが表示されます。このボタンをクリックすると、describe パラメータを使用して関数が呼び出され、関数のドキュメントが返されます。ドキュメントが Markdown 形式で適切にフォーマットされている場合、CloudWatch は YAML の 3 つの単一のバックティック文字 (```) で囲まれたドキュメントの最初のエントリを解析します。その後、パラメータ内のドキュメントが自動的に入力されます。以下は、そのように適切にフォーマットされたドキュメントの例です。

```
``` yaml
echo: <h1>Hello world</h1>
```
```

セキュリティと JavaScript

セキュリティ上の理由から、返される HTML では JavaScript を使用できません。JavaScript を削除することで、Lambda 関数の記述者によって、ダッシュボードでウィジェットを表示しているユーザーよりも高度なアクセス許可を使用して実行するコードを挿入される、アクセス許可のエスカレーションの問題を回避できます。

返された HTML に JavaScript コードまたはその他の既知のセキュリティの脆弱性が含まれている場合は、ダッシュボードにレンダリングされる前に HTML から消去されます。例えば、<iframe> タグおよび <use> タグは許可されず、削除されます。

カスタムウィジェットは、ダッシュボードではデフォルトでは実行されません。代わりに、カスタムウィジェットが呼び出す Lambda 関数を信頼する場合は、カスタムウィジェットを実行することを明示的に許可する必要があります。個々のウィジェットとダッシュボード全体の両方で、一度許可するか、常に許可するかを選択できます。個々のウィジェットとダッシュボード全体のアクセス許可を拒否することもできます。

カスタムウィジェットのインタラクティブ性

JavaScript が許可されていなくても、返された HTML とのインタラクティブ性を許可する他の方法があります。

- 返された HTML 内のすべての要素は、<cwdb-action> タグの特別な設定でタグ付けすることができ、その要素が選択されたときに、ポップアップで情報を表示したり、クリック時に確認を求めたり、任意の Lambda 関数を呼び出したりすることができます。例えば、Lambda 関数を使用して任意の AWS API を呼び出すボタンを定義できます。返される HTML は、既存の Lambda ウィジェットのコンテンツを置き換えるか、モーダル内に表示するように設定できます。
- 返される HTML には、新しいコンソールを開いたり、他の顧客のページを開いたり、他のダッシュボードを読み込んだりするためのリンクを含めることができます。
- HTML には、要素の `title` 属性を含めることができます。これにより、ユーザーがその要素の上にカーソルを置いた場合に追加情報が提供されます。
- 要素には、アニメーションやその他の CSS 効果を呼び出すことができる `:hover` などの CSS セレクターを含めることができます。ページ内の要素を表示または非表示にすることもできます。

<cwdb-action> 定義と使用

`<cwdb-action>` 要素は、直前の要素の動作を定義します。`<cwdb-action>` の内容は、表示する HTML か、Lambda 関数に渡すパラメータの JSON ブロックのいずれかです。

`<cwdb-action>` 要素の例を次に示します。

```
<cwdb-action
     action="call|html"
     confirmation="message"
     display="popup|widget"
     endpoint="<lambda ARN>"
     event="click|dblclick|mouseenter">

     html | params in JSON
</cwdb-action>
```

- action — 有効な値は、Lambda 関数を呼び出す `call` と、`<cwdb-action>` に含まれるすべての HTML を表示する `html` です。デフォルト: `html`。
- confirmation — 把握する必要のあるアクションを実行する前に確認メッセージを表示し、お客様がキャンセルできるようにします。

- display — 有効な値は popup および widget で、ウィジェット自体の内容を置き換えます。デフォルト: widget。
- endpoint - 呼び出す Lambda 関数の Amazon リソースネーム (ARN)。これは、action が call である場合に必要です。
- event — アクションを呼び出す前の要素のイベントを定義します。有効な値は、click、dblclick、mouseenter です。mouseenter イベントは、html アクションと組み合わせてしか使えません。デフォルト: click。

例

以下の例では、<cwdb-action> タグを使用して、Lambda 関数呼び出しによって Amazon EC2 インスタンスを再起動するボタンを作成する方法を示します。ここでは、通話の成功または失敗がポップアップで表示されます。

```
<a class="btn">Reboot Instance</a>
<cwdb-action action="call" endpoint="arn:aws:lambda:us-
east-1:123456:function:rebootInstance" display="popup">
       { "instanceId": "i-342389adbfef" }
</cwdb-action>
```

次の例では、ポップアップに詳細情報を表示します。

```
<a>Click me for more info in popup</a>
<cwdb-action display="popup">
   <h1>Big title</h1>
   More info about <b>something important</b>.
</cwdb-action>
```

この例では、[Next] (次へ) ボタンを使用して、ウィジェットのコンテンツを Lambda 関数への呼び出しに置き換えます。

```
<a class="btn btn-primary">Next</a>
<cwdb-action action="call" endpoint="arn:aws:lambda:us-
east-1:123456:function:nextPage">
   { "pageNum": 2 }
</cwdb-action>
```

カスタムウィジェットを作成する

カスタムウィジェットを作成するには、AWS が提供しているサンプルを使用するか、独自のサンプルを作成します。AWS のサンプルには、JavaScript と Python の両方のサンプルが含まれており、AWS CloudFormation スタックによって作成されています。サンプルの一覧については、「カスタムウィジェットのサンプル」を参照してください。

CloudWatch ダッシュボードでカスタムウィジェットを作成するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで [Dashboards] (ダッシュボード) を選択し、ダッシュボードをクリックします。
3. [+] 記号を選択します。
4. [Custom widget] (カスタムウィジェット) を選択します。
5. 次のいずれかの方法を使用します。
   - AWS が提供するカスタムウィジェットのサンプルを使用するには、次を実行します。
     a. ドロップダウンボックスでサンプルを選択します。

        AWS CloudFormation コンソールが新規のブラウザで起動します。AWS CloudFormation コンソールで、次の操作を行います。
     b. (オプション) AWS CloudFormation スタックの名前をカスタマイズします。
     c. サンプルで使用されるすべてのパラメータを選択します。
     d. [I acknowledge that AWS CloudFormation might create IAM resources] (AWS CloudFormation によって IAM リソースが作成される場合があることを承認します。) を選択し、[Create stack] (スタックの作成) を選択します。
   - AWS が提供する独自のカスタムウィジェットを作成するには、次の手順に従います。
     a. [Next] を選択します。
     b. Lambda 関数をリストから選択するか、Amazon リソースネーム (ARN) を入力するかを選択します。リストから選択する場合は、関数のあるリージョンと使用するバージョンも指定します。
     c. [Parameters] (パラメータ) で、関数で使用されるすべてのパラメータを選択します。
     d. ウィジェットのタイトルを入力します。
     e. [Update on] (更新日) で、ウィジェットを更新するタイミング (Lambda 関数を再度呼び出すタイミング) を設定します。これは、ダッシュボードの自動更新時にウィジェット

を更新するための [Refresh] (更新)、ウィジェットのサイズが変更されるたびに更新するための [Resize] (サイズ変更)、またはグラフのズームイン時などにダッシュボードの時間範囲が調整されるたびにウィジェットを更新する [Time Range] (時間範囲) のいずれかです。

f. プレビューに問題がなければ、[Create widget] (ウィジェットの作成) を選択します。

カスタムウィジェットのサンプル

AWS は、JavaScript と Python の両方でカスタムウィジェットのサンプルを提供します。これらのウィジェットのサンプルは、このリスト内の各ウィジェットのリンクを使用して作成できます。また、CloudWatch コンソールを使用してウィジェットを作成し、カスタマイズすることもできます。このリストのリンクから AWS CloudFormation コンソールを開き、[AWS CloudFormation quick-create] のリンクを使用してカスタムウィジェットを作成します。

GitHub で、カスタムウィジェットのサンプルにアクセスすることもできます。

このリストに続いて、言語ごとの Echo ウィジェットの完全な例が示されています。

JavaScript

JavaScript のカスタムウィジェットのサンプル

- Echo — 新しいウィジェットを作成することなく、カスタムウィジェットで HTML がどのように表示されるかをテストするために使用できる基本的な Echoer です。
- Hello world — 非常にベーシックなスターターウィジェットです。
- Custom widget debugger — Lambda ランタイム環境に関する有用な情報を表示するデバッガーウィジェットです。
- Query CloudWatch Logs Insights — CloudWatch Logs Insights でクエリを実行および編集します。
- Run Amazon Athena queries — Athena でクエリを実行および編集します。
- Call AWS API — 任意の読み取り専用 AWS API を呼び出し、結果を JSON 形式で表示します。
- Fast CloudWatch bitmap graph — サーバー側を使用して CloudWatch グラフをレンダリングし、高速表示します。
- Text widget from CloudWatch dashboard — 指定した CloudWatch ダッシュボードの最初のテキストウィジェットを表示します。

- CloudWatch metric data as a table — CloudWatch の生のメトリクスデータをテーブルに表示します。
- Amazon EC2 table — CPU 使用率が上位の EC2 インスタンスを表示します。このウィジェットには、デフォルトで無効になっている [再起動] ボタンも含まれます。
- AWS CodeDeploy デプロイ – CodeDeploy デプロイを表示します。
- AWS Cost Explorer レポート — 選択した時間範囲における AWS の各サービスのコストに関するレポートを表示します。
- Display content of external URL — 外部からアクセス可能な URL の内容を表示します。
- Display an Amazon S3 object — アカウントの Amazon S3 バケットにあるオブジェクトを表示します。
- Simple SVG pie chart — グラフィカルな SVG ベースのウィジェットの例です。

Python

Python のカスタムウィジェットのサンプル

- Echo - 新しいウィジェットを作成することなく、カスタムウィジェットで HTML がどのように表示されるかをテストするために使用できる基本的な Echoer です。
- Hello world — 非常にベーシックなスターターウィジェットです。
- Custom widget debugger — Lambda ランタイム環境に関する有用な情報を表示するデバッガーウィジェットです。
- Call AWS API — 任意の読み取り専用 AWS API を呼び出し、結果を JSON 形式で表示します。
- Fast CloudWatch bitmap graph — サーバー側を使用して CloudWatch グラフをレンダリングし、高速表示します。
- Send dashboard snapshot by email - 現在のダッシュボードのスナップショットを作成し、E メールの受信者に送信します。
- Send dashboard snapshot to Amazon S3 — 現在のダッシュボードのスナップショットを作成し、Amazon S3 に保存します。
- Text widget from CloudWatch dashboard — 指定した CloudWatch ダッシュボードの最初のテキストウィジェットを表示します。
- Display content of external URL — 外部からアクセス可能な URL の内容を表示します。
- RSS reader — RSS フィードを表示します。

- Display an Amazon S3 object — アカウントの Amazon S3 バケットにあるオブジェクトを表示します。
- Simple SVG pie chart — グラフィカルな SVG ベースのウィジェットの例です。

JavaScript での Echo ウィジェット

以下は、JavaScript での Echo ウィジェットのサンプルです。

```
const DOCS = `
## Echo
A basic echo script. Anything passed in the \`\`\`echo\`\`\` parameter is returned as
 the content of the custom widget.
### Widget parameters
Param | Description
---|---
**echo** | The content to echo back

### Example parameters
\`\`\` yaml
echo: <h1>Hello world</h1>
\`\`\`
`;

exports.handler = async (event) => {
    if (event.describe) {
        return DOCS;
    }

    let widgetContext = JSON.stringify(event.widgetContext, null, 4);
    widgetContext = widgetContext.replace(/</g, '&lt;');
    widgetContext = widgetContext.replace(/>/g, '&gt;');

    return `${event.echo || ''}<pre>${widgetContext}</pre>`;
};
```

Python での Echo ウィジェット

以下は、Python での Echo ウィジェットのサンプルです。

```
import json

DOCS = """
```

```
## Echo
A basic echo script. Anything passed in the ```echo``` parameter is returned as the
 content of the custom widget.
### Widget parameters
Param | Description
---|---
**echo** | The content to echo back

### Example parameters
``` yaml
echo: <h1>Hello world</h1>
```"""

def lambda_handler(event, context):
    if 'describe' in event:
        return DOCS

    echo = event.get('echo', '')
    widgetContext = event.get('widgetContext')
    widgetContext = json.dumps(widgetContext, indent=4)
    widgetContext = widgetContext.replace('<', '<')
    widgetContext = widgetContext.replace('>', '>')

    return f'{echo}<pre>{widgetContext}</pre>'
```

Java での Echo ウィジェット

以下は、Java での Echo ウィジェットのサンプルです。

```
package example;

import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.RequestHandler;

import com.google.gson.Gson;
import com.google.gson.GsonBuilder;

public class Handler implements RequestHandler<Event, String>{

  static String DOCS = ""
    + "## Echo\n"
    + "A basic echo script. Anything passed in the ```echo``` parameter is returned as
 the content of the custom widget.\n"
    + "### Widget parameters\n"
```

```
    + "Param | Description\n"
    + "---|---\n"
    + "**echo** | The content to echo back\n\n"
    + "### Example parameters\n"
    + "```yaml\n"
    + "echo: <h1>Hello world</h1>\n"
    + "```\n";

  Gson gson = new GsonBuilder().setPrettyPrinting().create();

  @Override
  public String handleRequest(Event event, Context context) {

    if (event.describe) {
      return DOCS;
    }

    return (event.echo != null ? event.echo : "") + "<pre>" +
 gson.toJson(event.widgetContext) + "</pre>";
  }
}

class Event {

    public boolean describe;
    public String echo;
    public Object widgetContext;

    public Event() {}

    public Event(String echo, boolean describe, Object widgetContext) {
        this.describe = describe;
        this.echo = echo;
        this.widgetContext = widgetContext;
    }
}
```

CloudWatch ダッシュボードからテキストウィジェットを追加または削除する

テキストウィジェットにはマークダウン形式でテキストブロックが含まれます。CloudWatch ダッシュボードを使用して、テキストウィジェットの追加、編集、削除を行うことができます。

テキストのウィジェットをダッシュボードに追加する方法

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで [Dashboards] (ダッシュボード) を選択し、ダッシュボードをクリックします。
3. [+] 記号を選択します。
4. [テキスト] を選択します。
5. [マークダウン] で、[マークダウン] を使用してテキストを追加してフォーマットし、[ウィジェットの作成] を選択します。
6. テキストウィジェットを透明にするには、[透明な背景] を選択します。
7. [ダッシュボードの保存] を選択します。

ダッシュボードのテキストウィジェットを編集する方法

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで [Dashboards] (ダッシュボード) を選択し、ダッシュボードをクリックします。
3. テキストブロックの右上隅にカーソルを合わせ、[Widget actions] (ウィジェットのアクション) をクリックします。続いて、[Edit] (編集) をクリックします。
4. 必要に応じてテキストを更新し、[ウィジェットの更新] を選択します。
5. [ダッシュボードの保存] を選択します。

テキストウィジェットをダッシュボードから削除する方法

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで [Dashboards] (ダッシュボード) を選択し、ダッシュボードをクリックします。
3. テキストブロックの右上隅にカーソルを合わせ、[Widget actions] (ウィジェットのアクション) をクリックします。その後、[Delete] (削除) をクリックします。
4. [ダッシュボードの保存] を選択します。

CloudWatch ダッシュボードでアラームウィジェットを追加または削除する

アラームウィジェットをダッシュボードに追加するには、以下のオプションのいずれかを選択します。

- アラームメトリクスのグラフとアラームステータスを表示する 1 つのアラームをウィジェットに追加します。
- 複数のアラームステータスをグリッドで表示するには、アラームステータスウィジェットを追加します。アラーム名と現在のステータスのみが表示され、グラフは表示されません。1 つのアラームステータスウィジェットには、最大 100 個のアラームを含めることができます。

グラフを含む 1 つのアラームをダッシュボードに追加するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで、[アラーム] を選択し、追加するアラームを選択したら、[ダッシュボードに追加] をクリックします。
3. ダッシュボードを選択し、ウィジェットの種類 ([ライン]、[スタックされたエリア]、[数値] のいずれか) を選択して、[ダッシュボードに追加] を選択します。
4. ダッシュボードのアラームを表示するには、ナビゲーションペインの [ダッシュボード] を選択し、ダッシュボードを選択します。
5. (オプション) アラームのグラフを一時的に大きくするには、グラフを選択します。
6. (オプション) ウィジェットタイプを変更するには、グラフのタイトルの上にマウスを移動し、[Widget actions (ウィジェットのアクション)]、[Widget type (ウィジェットタイプ)] の順に選択します。

アラームステータスウィジェットをダッシュボードに追加するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで [Dashboards] (ダッシュボード) を選択し、ダッシュボードをクリックします。
3. [+] 記号を選択します。
4. [Alarm status (アラームステータス)] を選択します。
5. ウィジェットに追加するアラームの横にあるチェックボックスをオンにし、[Create widget (ウィジェットの作成)] を選択します。

6. [ダッシュボードに追加] を選択します。

ダッシュボードからアラームウィジェットを削除するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで [Dashboards] (ダッシュボード) を選択し、ダッシュボードをクリックします。
3. ウィジェットにカーソルを合わせて、[Widget actions (ウィジェットのアクション)] を選択し、[Delete (削除)] を選択します。
4. [ダッシュボードの保存] を選択します。変更を保存する前にダッシュボードから離れようとすると、変更を保存するか破棄するよう求められます。

CloudWatch ダッシュボードからのデータテーブルウィジェットの追加または削除

データテーブルウィジェットを使用すると、未加工データポイントのメトリクスと、未加工データについて簡単にまとめた情報を参照できます。データテーブルウィジェットは実際のデータを抽象化したチャートではないため、表示されているデータポイントの理解が容易になります。ここで示す手順では、CloudWatch ダッシュボードでデータテーブルウィジェットを追加または削除する方法について説明します。

テーブルプロパティ

データテーブルにはデフォルトのプロパティセットがあるため、オプションやソースに変更を加える必要はありません。スティッキーラベル列、有効になっているすべての集計列、四捨五入されたデータポイント、その換算単位といったプロパティがあります。

各データテーブルウィジェットには、以下のプロパティを含めることができます。プロパティごとに、ダッシュボードの JSON ソースでプロパティを設定するにはどうすればよいかといった情報があります。ダッシュボード JSON ソースの詳細については、「ダッシュボード本体の構造と構文」を参照してください。

[概要]

集計列は、データテーブルウィジェットに導入された新しいプロパティです。現在のテーブルのうち、ある特定の部分を集計したものです。例えば、Sum 集計はその行に表示されているすべてのデータポイントの合計です。集計列は、CloudWatch 統計とは異なります。ソースでは、次のようになっています。

```
 "table": {
        "summaryColumns": [
            "MIN",
            "MAX",
            "SUM",
            "AVG"
        ]
    },
```

しきい値

テーブルにしきい値を適用するために使用します。データポイントがしきい値内になると、対応するセルがそのしきい値の色で強調表示されます。ソースでは、次のようになっています。

```
"annotations": {
    "horizontal": [
        {
            "label": string,
            "value": int,
            "fill": "above" | "below"
        }
    ]
}
```

ラベル列の単位

メトリクスに関連付けられている単位を表示する場合は、このオプションを有効にすると、ラベルの横にあるラベル列に単位が表示されます。ソースでは、次のようになっています。

```
"yAxis": {
```

```
    "left": {
        "showUnits": true | false
    }
}
```

行と列の反転

テーブルが変換されて、データポイントが列から行に切り替わり、メトリクスが列になります。ソースでは、次のようになっています。

```
 "table": {
    "layout": "vertical" | "horizontal"
}
```

スティッキー集計列

集計列が固定され、スクロールしても表示されたままになります。ラベルは既に固定されています。ソースでは、次のようになっています。

```
"table": {
    "stickySummary": true | false
}
```

集計列のみの表示

データポイント列を非表示にして、ラベル列と集計列のみを表示できます。ソースでは、次のようになっています。

```
 "table": {
    "showTimeSeriesData": false | true
}
```

ライブデータ

集計がまだ完了していなくても、最新のデータポイントを表示します。ソースでは、次のようになっています。

```
"liveData": true | false
```

数値ウィジェットの形式

四捨五入や換算の前に、セルに収まる分だけの数値を表示します。ソースでは、次のようになっています。

```
"singleValueFullPrecision": true | false
```

データテーブルウィジェットをダッシュボードに追加するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで [ダッシュボード] を選択し、ダッシュボードをクリックします。
3. [+] ボタンを選択し、[データテーブル] を選択して [次へ] を選択します。
4. [参照] タブで、テーブルウィジェットに表示するメトリクスを検索または参照します。次に、メトリクスを選択します。
5. (オプション) テーブルのレイアウトを変更するには、[オプション] タブを選択し、[行と列を反転] を選択します。

   また、[オプション] タブを使用すると、テーブルに表示される列を変更できるほか、[ラベル] 列に使用されている単位を表示することもできます。

   > **Tip**
   >
   > 表示するしきい値の精度を高めるには、[四捨五入する前に最大表示可能桁数を表示する] を選択します。

6. (オプション) データテーブルウィジェットの時間範囲を変更するには、ウィジェットの上部にある事前定義済みの時間範囲を選択します。1 時間から 1 週間の範囲で選択します。独自の時間範囲を設定するには、[Custom] (カスタム) をクリックします。
7. (オプション) データテーブルウィジェットの時間範囲を変更するには、ウィジェットの上部にある事前定義済みの時間範囲を選択します。1 時間から 1 週間の範囲で選択します。独自の時間範囲を設定するには、[Custom] (カスタム) をクリックします。
8. (オプション) 後でダッシュボードの他の部分の時間範囲が変更されても、選択した時間範囲が引き続きウィジェットで使用されるようにするには、[持続時間範囲] を選択します。
9. [ウィジェットの作成]、[ダッシュボードの保存] の順にクリックします。

ダッシュボードからテーブルウィジェットを削除するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで [Dashboards] (ダッシュボード) を選択し、ダッシュボードをクリックします。
3. 削除するウィジェットの右上隅で、[ウィジェットのアクション]、[削除] の順にクリックします。
4. [ダッシュボードの保存] を選択します。

CloudWatch ダッシュボードでグラフをリンクおよびリンク解除する

ダッシュボード上のグラフをリンクし、1 つのグラフを拡大または縮小した際に同時に他のグラフも縮小または拡大されるようにできます。グラフのリンクを解除し、拡大を 1 つのグラフに行うこともできます。

ダッシュボードのグラフのリンクする方法

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで [Dashboards] (ダッシュボード) を選択し、ダッシュボードをクリックします。
3. [アクション]、[グラフのリンク] の順に選択します。

ダッシュボードのグラフのリンクを解除するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで [Dashboards] (ダッシュボード) を選択し、ダッシュボードをクリックします。
3. [アクション]、[グラフのリンク] をオフにします。

CloudWatch ダッシュボードの共有

CloudWatch ダッシュボードは、AWS アカウントに直接アクセスできないユーザーと共有できます。これにより、チーム間ではもちろん、利害関係者や組織以外の相手ともダッシュボードを共有できます。チームエリアの大画面にダッシュボードを表示したり、Wiki やその他のウェブページに組み込んだりすることもできます。

> **Warning**
>
> ダッシュボードを共有するすべてのユーザーに対し、アカウントの ダッシュボードを共有しているユーザーに付与されるアクセス許可 に一覧表示されているアクセス許可が付与されます。ダッシュボードをパブリックに共有する場合は、ダッシュボードへのリンクを持つすべてのユーザーに対し、これらの許可を付与することになります。
>
> `cloudwatch:GetMetricData` および `ec2:DescribeTags` アクセス許可は、特定のメトリクスまたは EC2 インスタンスに対しスコープダウンすることはできません。つまり、ダッシュボードへのアクセス権を持つユーザーは、アカウント内のすべての CloudWatch メトリクス、およびすべての EC2 インスタンスの名前とタグに対してクエリできます。

ダッシュボードを共有する場合、ダッシュボードを表示できるユーザーを 3 つの方法で指定できます。

- 1 つのダッシュボードを共有し、ダッシュボードを表示できるユーザーに与えられた E メールアドレスを 5 つまで指定できます。これらのユーザーはそれぞれ独自のパスワードを作成し、ダッシュボードを表示するときにそのパスワードを入力する必要があります。
- 1 つのダッシュボードをパブリックに共有し、リンクを持っている全員がダッシュボードを表示できるようにします。
- アカウントのすべての CloudWatch ダッシュボードを共有し、ダッシュボードアクセス用のサードパーティーのシングルサインオン (SSO) プロバイダーを指定します。この SSO プロバイダー一覧のメンバーであるすべてのユーザーは、アカウントのすべてのダッシュボードにアクセスできます。これを有効にするには、SSO プロバイダーを Amazon Cognito と統合します。SSO プロバイダーは、Security Assertion Markup Language (SAML) をサポートしている必要があります。Amazon Cognito の詳細については、「Amazon Cognito とは」を参照してください。

ダッシュボードを共有しても料金は発生しませんが、共有ダッシュボード内のウィジェットには標準の CloudWatch 料金がかかります。CloudWatch の料金の詳細については、Amazon CloudWatch の料金をご覧ください。

ダッシュボードを共有すると、Amazon Cognito リソースが米国東部 (バージニア北部) リージョンに作成されます。

> **Important**
>
> ダッシュボード共有プロセスで作成されたリソースの名前と識別子は変更しないでください。これには、Amazon Cognito と IAM のリソースが含まれます。これらのリソースを変更すると、共有ダッシュボードが予期せず正しく機能しなくなる可能性があります。

> **Note**
>
> アラーム注釈付きのメトリクスウィジェットを持つダッシュボードを共有する場合、ダッシュボードを共有するユーザーにはこれらのウィジェットは表示されません。代わりに、ウィジェットを利用できないことを示すテキストを含む空白のウィジェットが表示されます。ダッシュボードを自分で表示すると、アラーム注釈付きのメトリクスウィジェットが表示されます。

ダッシュボードを共有するために必要な許可

以下のいずれかの方法を使用してダッシュボードを共有したり、どのダッシュボードが共有済みであるかを確認したりするには、ユーザーとしてサインオンするか、特定のアクセス許可のある IAM ロールが必要です。

ダッシュボードを共有できるようにするには、ユーザーまたは IAM ロールに、以下のポリシーステートメント内のアクションを含める必要があります。

```
{
    "Effect": "Allow",
    "Action": [
        "iam:CreateRole",
        "iam:CreatePolicy",
        "iam:AttachRolePolicy",
        "iam:PassRole"
    ],
    "Resource": [
        "arn:aws:iam::*:role/service-role/CWDBSharing*",
        "arn:aws:iam::*:policy/*"
    ]
},
{
```

```
    "Effect": "Allow",
    "Action": [
        "cognito-idp:*",
        "cognito-identity:*",
    ],
    "Resource": [
        "*"
    ]
},
{
    "Effect": "Allow",
    "Action": [
        "cloudwatch:GetDashboard",
    ],
    "Resource": [
        "*"
        // or the ARNs of dashboards that you want to share
    ]
}
```

共有されているダッシュボードを確認できるものの、ダッシュボードを共有できないようにするには、ユーザーまたは IAM ロールに以下のポリシーステートメントを含めることができます。

```
{
    "Effect": "Allow",
    "Action": [
        "cognito-idp:*",
        "cognito-identity:*"
    ],
    "Resource": [
        "*"
    ]
},
{
    "Effect": "Allow",
    "Action": [
        "cloudwatch:ListDashboards",
    ],
    "Resource": [
        "*"
    ]
}
```

ダッシュボードを共有しているユーザーに付与されるアクセス許可

ダッシュボードを共有すると、対象のアカウントに CloudWatch が IAM ロールを作成します。このロールにより、ダッシュボードを共有するユーザーに次のアクセス許可が付与されます。

- `cloudwatch:GetInsightRuleReport`
- `cloudwatch:GetMetricData`
- `cloudwatch:DescribeAlarms`
- `ec2:DescribeTags`

> **Warning**
>
> ダッシュボードを共有するすべてのユーザーに対し、アカウントに対するこれらのアクセス許可が付与されます。ダッシュボードをパブリックに共有する場合は、ダッシュボードへのリンクを持つすべてのユーザーに対し、これらの許可を付与することになります。`cloudwatch:GetMetricData` および `ec2:DescribeTags` アクセス許可は、特定のメトリクスまたは EC2 インスタンスに対しスコープダウンすることはできません。つまり、ダッシュボードへのアクセス権を持つユーザーは、アカウント内のすべての CloudWatch メトリクス、およびすべての EC2 インスタンスの名前とタグに対してクエリできます。

ダッシュボードを共有する場合、デフォルトでは、CloudWatch が作成するアクセス許可によって、ダッシュボードが共有されるときにダッシュボードにあるアラームおよび Contributor Insights ルールのみへのアクセスが制限されます。新しいアラームまたは Contributor Insights ルールをダッシュボードに追加し、ダッシュボードを共有したユーザーにも表示する場合は、これらのリソースを許可するようにポリシーを更新する必要があります。

特定のユーザーとの 1 つのダッシュボードの共有

選択した最大 5 つの E メールアドレスとダッシュボードを共有するには、このセクションのステップに従います。

> **Note**
>
> デフォルトでは、ダッシュボード上の CloudWatch Logs ウィジェットは、ダッシュボードを共有しているユーザーには表示されません。詳細については、「共有しているユーザーが、ログテーブルウィジェットを表示できるようにする」を参照してください。

デフォルトでは、ダッシュボード上の複合アラームウィジェットは、ダッシュボードを共有しているユーザーには表示されません。詳細については、「共有しているユーザーが複合アラームを表示できるよう許可する」を参照してください。

特定のユーザーとダッシュボードを共有するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで、ダッシュボードを選択します。
3. ダッシュボードの名前を選択します。
4. [Actions (アクション)]、[Share dashboard (ダッシュボードの共有)] の順に選択します。
5. [Share your dashboard and require a username and password (ダッシュボードを共有し、ユーザー名とパスワードを要求する)] の横にある [Start sharing (共有を開始)] を選択します。
6. [Add email addresses (E メールアドレスの追加)] で、ダッシュボードを共有する E メールアドレスを入力します。最大 5 つの E メールアドレスを含めることができます。
7. すべての E メールアドレスを入力したら、契約書を読み、確認ボックスを選択します。次に [Preview policy] (ポリシーのプレビュー) を選択します。
8. 共有するリソースが必要なものであることを確認し、[Confirm and generate shareable link] (共有可能なリンクを確認して生成する) を選択します。
9. 次のページで、[Copy link to clipboard (リンクをクリップボードにコピー)] を選択します。その後、このリンクを E メールに貼り付けて、招待ユーザーに送信できます。ダッシュボードへの接続に使用するユーザー名と一時パスワードが記載された個別の E メールが、それらのユーザーに自動的に送信されます。

1 つのダッシュボードのパブリック共有

ダッシュボードをパブリックに共有するには、このセクションのステップに従います。これは、ダッシュボードをチームルームの大画面に表示したり、Wiki ページに埋め込んだりする場合に便利です。

Important

ダッシュボードをパブリックに共有すると、リンクを持つすべてのユーザーが認証なしでダッシュボードにアクセスできるようになります。これは、機密情報が含まれていないダッシュボードに対してのみ行ってください。

> **Note**
>
> デフォルトでは、ダッシュボード上の CloudWatch Logs ウィジェットは、ダッシュボードを共有しているユーザーには表示されません。詳細については、「共有しているユーザーが、ログテーブルウィジェットを表示できるようにする」を参照してください。
> デフォルトでは、ダッシュボード上の複合アラームウィジェットは、ダッシュボードを共有しているユーザーには表示されません。詳細については、「共有しているユーザーが複合アラームを表示できるよう許可する」を参照してください。

ダッシュボードをパブリックに共有するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで、ダッシュボードを選択します。
3. ダッシュボードの名前を選択します。
4. [Actions (アクション)]、[Share dashboard (ダッシュボードの共有)] の順に選択します。
5. [Share your dashboard publicly (ダッシュボードをパブリックに共有)] の横にある [Start sharing (共有を開始)] を選択します。
6. テキストボックスに「**`Confirm`**」と入力します。
7. 契約書を読み、確認ボックスを選択します。次に [Preview policy] (ポリシーのプレビュー) を選択します。
8. 共有するリソースが必要なものであることを確認し、[Confirm and generate shareable link] (共有可能なリンクを確認して生成する) を選択します。
9. 次のページで、[Copy link to clipboard (リンクをクリップボードにコピー)] を選択します。その後、このリンクを共有できます。リンクを共有するユーザーは、資格情報を入力せずにダッシュボードにアクセスできます。

SSO を使用してアカウント内のすべての CloudWatch ダッシュボードを共有する

シングルサインオン (SSO) を使用してアカウント内のすべてのダッシュボードをユーザーと共有するには、このセクションのステップを使用します。

> **Note**
>
> デフォルトでは、ダッシュボード上の CloudWatch Logs ウィジェットは、ダッシュボードを共有しているユーザーには表示されません。詳細については、「共有しているユーザーが、ログテーブルウィジェットを表示できるようにする」を参照してください。
> デフォルトでは、ダッシュボード上の複合アラームウィジェットは、ダッシュボードを共有しているユーザーには表示されません。詳細については、「共有しているユーザーが複合アラームを表示できるよう許可する」を参照してください。

SSO プロバイダーのリストに含まれているユーザーと CloudWatch ダッシュボードを共有するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで、ダッシュボードを選択します。
3. ダッシュボードの名前を選択します。
4. [Actions (アクション)]、[Share dashboard (ダッシュボードの共有)] の順に選択します。
5. [Go to CloudWatch Settings (CloudWatch 設定に移動)] を選択します。
6. 目的の SSO プロバイダーが [Available SSO providers (使用可能な SSO プロバイダー)] に含まれていない場合、[Manage SSO providers (SSO プロバイダーの管理)] を選択し、「CloudWatch ダッシュボード共有用の SSO を設定する」の手順に従います。

   次に、CloudWatch コンソールに戻り、ブラウザを更新します。有効にした SSO プロバイダーがリストに表示されます。
7. [Available SSO providers (使用可能な SSO プロバイダー)] リストで、目的の SSO プロバイダーを選択します。
8. [Save changes] (変更の保存) をクリックします。

CloudWatch ダッシュボード共有用の SSO を設定する

SAML をサポートするサードパーティーのシングルサインオンプロバイダーを使用してダッシュボード共有を設定するには、次のステップを実行します。

> **Important**
>
> 非 SAML SSO プロバイダーを使用してダッシュボードを共有しないことを強くお勧めします。この方法で共有すると、誤ってサードパーティーがアカウントのダッシュボードにアクセスできるようになるリスクがあります。

ダッシュボード共有を有効にするために SSO プロバイダーを設定するには

1. SSO プロバイダーを Amazon Cognito と統合します。詳細については、「サードパーティーの SAML ID プロバイダーと Amazon Cognito ユーザープールの統合」を参照してください。
2. SSO プロバイダからメタデータ XML ファイルをダウンロードします。
3. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
4. ナビゲーションペインで [設定] を選択します。
5. [Dashboard sharing (ダッシュボード共有)] セクションで、[Configure (設定)] を選択します。
6. [Manage SSO providers (SSO プロバイダーの管理)] を選択します。

   これにより、Amazon Cognito コンソールが米国東部 (バージニア北部) リージョン (us-east-1) で開きます。ユーザープールが何も表示されない場合は、Amazon Cognito コンソールが別のリージョンで開かれている可能性があります。その場合は、リージョンを米国東部 (バージニア北部) us-east-1 に変更し、次の手順に進みます。
7. [CloudWatchDashboardSharing] プールを選択します。
8. ナビゲーションペインで、[Identity providers (ID プロバイダー)] を選択します。
9. SAML を選択します。
10. [Provider name (プロバイダー名)] に SSO プロバイダーの名前を入力します。
11. [Select file (ファイルの選択)] を選択し、ステップ 1 でダウンロードしたメタデータ XML ファイルを選択します。
12. [プロバイダーの作成] を選択します。

共有されているダッシュボードの数を確認する

CloudWatch コンソールを使用して、現在他のユーザーと共有されている CloudWatch ダッシュボードの数を確認できます。

共有されているダッシュボードの数を確認するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで [設定] を選択します。
3. [Dashboard sharing] (ダッシュボードの共有) セクションには、共有されているダッシュボードの数が表示されます。
4. 共有されているダッシュボードを確認するには、[Username and password] (ユーザー名とパスワード) および [Public dashboards] (パブリックダッシュボード) の [***number*** dashboards shared] (共有されているダッシュボードの数) を選択します。

どのダッシュボードが共有されているかを確認する

CloudWatch コンソールを使用して、現在他のユーザーと共有されているダッシュボードを確認できます。

共有されているダッシュボードを確認するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで、ダッシュボードを選択します。
3. ダッシュボードのリストで、[共有] 列を確認します。この列にアイコンが入力されているダッシュボードは、現在共有されています。
4. ダッシュボードを共有しているユーザーを確認するには、ダッシュボード名を選択し、[アクション]、[ダッシュボードの共有] の順に選択します。

   共有ダッシュボードの**########**ページには、ダッシュボードの共有方法が表示されます。必要に応じて、[共有の停止] を選択して、ダッシュボードの共有を停止できます。

1 つ以上のダッシュボードの共有を停止する

単一の共有ダッシュボードの共有を停止することも、すべての共有ダッシュボードの共有を一度に停止することもできます。

1 つのダッシュボードの共有を停止するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで、ダッシュボードを選択します。

3. 共有ダッシュボードの名前を選択します。
4. [Actions (アクション)]、[Share dashboard (ダッシュボードの共有)] の順に選択します。
5. [共有の停止] を選択します。
6. 確認ボックスで、[共有の停止] を選択します。

すべての共有ダッシュボードの共有を停止するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで [設定] を選択します。
3. [Dashboard sharing] (ダッシュボードの共有) セクションで、[Stop sharing all dashboards] (すべてのダッシュボードの共有を停止) を選択します。
4. 確認ボックスで、[Stop sharing all dashboards] (すべてのダッシュボードの共有の停止) を選択します。

共有ダッシュボードの許可を確認し、許可の範囲を変更する

共有ダッシュボードでのユーザーのアクセス許可を確認する場合、または共有ダッシュボードのアクセス許可の範囲を変更する場合は、このセクションに記載されている手順を使用してください。

共有ダッシュボードの許可を確認するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで、ダッシュボードを選択します。
3. 共有ダッシュボードの名前を選択します。
4. [Actions (アクション)]、[Share dashboard (ダッシュボードの共有)] の順に選択します。
5. [Resources (リソース)] で、[IAM Role (IAM ロール)] を選択します。
6. IAM コンソールで、表示されたポリシーを選択します。
7. (オプション) 共有ダッシュボードユーザーが表示できるアラームを制限するには、[Edit policy] (ポリシーの編集) を選択し、`cloudwatch:DescribeAlarms` 許可を現在の位置から共有ダッシュボードのユーザーに表示したいアラームのみの ARN を表示する新しい `Allow` ステートメントに移動します。次の例を参照してください。

```
{
   "Effect": "Allow",
    "Action": "cloudwatch:DescribeAlarms",
```

```
    "Resource": [
        "AlarmARN1",
        "AlarmARN2"
    ]
}
```

これを行う場合は、現在のポリシーの次のようなセクションから必ず cloudwatch:DescribeAlarms 許可を削除してください。

```
{
   "Effect": "Allow",
    "Action": [
        "cloudwatch:GetInsightRuleReport",
        "cloudwatch:GetMetricData",
        "cloudwatch:DescribeAlarms",
        "ec2:DescribeTags"
    ],
    "Resource": "*"
}
```

8. (オプション) 共有ダッシュボードユーザーが表示できる Contributor Insights ルールの範囲を制限するには、[ポリシーの編集] を選択し、 cloudwatch:GetInsightRuleReport を現在の位置から共有ダッシュボードユーザーに表示したい Contributor Insights ルールのみの ARN を表示する新しい Allow ステートメントに移動します。次の例を参照してください。

```
{
   "Effect": "Allow",
    "Action": "cloudwatch:GetInsightRuleReport",
    "Resource": [
        "PublicContributorInsightsRuleARN1",
        "PublicContributorInsightsRuleARN2"
    ]
}
```

これを行う場合は、現在のポリシーの次のようなセクションから必ず cloudwatch:GetInsightRuleReport を削除してください。

```
{
            "Effect": "Allow",
            "Action": [
                "cloudwatch:GetInsightRuleReport",
```

```
                "cloudwatch:GetMetricData",
                "cloudwatch:DescribeAlarms",
                "ec2:DescribeTags"
            ],
            "Resource": "*"
        }
```

共有しているユーザーが複合アラームを表示できるよう許可する

ダッシュボードを共有する場合、デフォルトでは、ダッシュボード上の複合アラームウィジェットは、ダッシュボードを共有しているユーザーには表示されません。複合アラームウィジェットを表示するには、ダッシュボード共有ポリシーに DescribeAlarms: * 許可を追加する必要があります。その許可は以下のようになります。

```
{
    "Effect": "Allow",
    "Action": "cloudwatch:DescribeAlarms",
    "Resource": "*"
}
```

Warning

前述のポリシーステートメントは、アカウント内のすべてのアラームへのアクセスを付与します。cloudwatch:DescribeAlarms の範囲を縮小するには、Deny ステートメントを使用する必要があります。ポリシーに Deny ステートメントを追加し、ロックダウンするアラームの ARM を指定できます。その拒否ステートメントは、次のようになります。

```
{
    "Effect": "Allow",
    "Action": "cloudwatch:DescribeAlarms",
    "Resource": "*"
},
{
    "Effect": "Deny",
    "Action": "cloudwatch:DescribeAlarms",
    "Resource": [
        "SensitiveAlarm1ARN",
        "SensitiveAlarm1ARN"
    ]
```

```
}
```

共有しているユーザーが、ログテーブルウィジェットを表示できるようにする

ダッシュボードを共有する場合、デフォルトでは、ダッシュボード上の CloudWatch Logs Insights ウィジェットは、ダッシュボードを共有しているユーザーには表示されません。これは、現在存在する CloudWatch Logs Insights ウィジェットと、共有後にダッシュボードに追加される Insights ウィジェットの両方に影響を与えます。

これらのユーザーが CloudWatch Logs ウィジェットを表示できるようにするには、ダッシュボードを共有するために IAM ロールにアクセス許可を追加する必要があります。

ダッシュボードを共有するユーザーに CloudWatch Logs ウィジェットを表示できるようにするには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで、ダッシュボードを選択します。
3. 共有ダッシュボードの名前を選択します。
4. [Actions (アクション)]、[Share dashboard (ダッシュボードの共有)] の順に選択します。
5. [Resources (リソース)] で、[IAM Role (IAM ロール)] を選択します。
6. IAM コンソールで、表示されたポリシーを選択します。
7. [Edit policy (ポリシーの編集)] を選択し、次のステートメントを追加します。新しいステートメントでは、共有するロググループのみの ARN を指定することをお勧めします。次の例を参照してください。

```
{
            "Effect": "Allow",
            "Action": [
                "logs:FilterLogEvents",
                "logs:StartQuery",
                "logs:StopQuery",
                "logs:GetLogRecord",
                "logs:DescribeLogGroups"
            ],
            "Resource": [
                "SharedLogGroup1ARN",
                "SharedLogGroup2ARN"
```

```
            ]
        },
```

8. [Save Changes] を選択します。

ダッシュボード共有のための IAM ポリシーに、リソースとして * を持つ 5 つのアクセス許可が既に含まれている場合は、ポリシーを変更し、共有するロググループの ARN のみを指定することを強くお勧めします。例えば、このようなアクセス許可の Resource セクションが以下だったとします。

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

次の例のように、共有するロググループの ARN だけを指定するようにポリシーを変更します。

```
"Resource": [
  "SharedLogGroup1ARN",
  "SharedLogGroup2ARN"
]
```

共有しているユーザーがカスタムウィジェットを表示できるようにする

ダッシュボードを共有する場合、デフォルトでは、ダッシュボード上のカスタムウィジェットは、ダッシュボードを共有しているユーザーには表示されません。これは、現存するカスタムウィジェットと、共有後にダッシュボードに追加されるカスタムウィジェットの両方に影響を与えます。

これらのユーザーがカスタムウィジェットを表示できるようにするには、IAM ロールにアクセス許可を追加してダッシュボードを共有する必要があります。

ダッシュボードを共有するユーザーがカスタムウィジェットを表示できるようにするには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで、ダッシュボードを選択します。
3. 共有ダッシュボードの名前を選択します。
4. [Actions (アクション)]、[Share dashboard (ダッシュボードの共有)] の順に選択します。
5. [Resources (リソース)] で、[IAM Role (IAM ロール)] を選択します。
6. IAM コンソールで、表示されたポリシーを選択します。
7. [Edit policy (ポリシーの編集)] を選択し、次のステートメントを追加します。新しいステートメントでは、共有する Lambda 関数のみの ARN を指定することをお勧めします。次の例を参照してください。

```
{
  "Sid": "Invoke",
  "Effect": "Allow",
  "Action": [
      "lambda:InvokeFunction"
  ],
  "Resource": [
    "LambdaFunction1ARN",
    "LambdaFunction2ARN"
  ]
 }
```

8. [Save Changes] を選択します。

ダッシュボード共有のための IAM ポリシーに、リソースとして * を持つアクセス許可が既に含まれている場合は、ポリシーを変更し、共有する Lambda 関数の ARN のみを指定することを強くお勧めします。例えば、このようなアクセス許可の Resource セクションが以下だったとします。

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

次の例のように、共有するカスタムウィジェットの ARN のみを指定するようにポリシーを変更します。

```
"Resource": [
  "LambdaFunction1ARN",
  "LambdaFunction2ARN"
]
```

ライブデータを使用する

メトリックウィジェットにライブデータを表示するかどうかを選択できます。ライブデータとは、直近の 1 分以内に公開された、完全には集計されていないデータです。

- ライブデータを オフにすると、集計期間が 1 分以上経過したデータポイントのみが表示されます。たとえば、5 分の期間を使用すると、12 時 35 分のデータポイントは 12 時 35 分から 12 時 40 分まで集計され、12 時 41 分に表示されます。
- ライブデータを オンにすると、対応する集計期間にデータが公開されるとすぐに、最新のデータポイントが表示されます。表示を更新するたびに、その集計期間内に新しいデータが公開される

と、最新のデータポイントが変わる場合があります。[合計] や [サンプル数] などの累積統計情報を使用する場合、ライブデータをオンにすると、グラフの最後に急減が生じることがあります。

ライブデータをダッシュボード全体で有効にするか、ダッシュボードの個別のウィジェットで有効にするかを選択できます。

ライブデータをダッシュボード全体で使用するかどうかを選択するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで [Dashboards] (ダッシュボード) を選択し、ダッシュボードをクリックします。
3. ライブデータをダッシュボードのすべてのウィジェットで永久的にオンまたはオフにするには、次の操作を行います。
   a. [アクション]、[設定]、[Bulk update live data] の順に選択します。
   b. [Live Data on] (ライブデータをオンにする) または [Live Data off] (ライブデータをオフにする) のいずれかを選択し、[Set] (設定) を選択します。
4. 各ウィジェットのライブデータ設定を一時的に上書きするには、[アクション] を選択します。次に、[ライブデータ] の横の [オーバーライド] で、次のいずれかの操作を行います。
   - すべてのウィジェットでライブデータを一時的にオンにするには、[On] を選択します。
   - すべてのウィジェットでライブデータを一時的にオフにするには、[Off] を選択します。
   - 各ウィジェットのライブデータ設定を保持するには、[Do not override] を選択します。

個別のウィジェットでライブデータを使用するかどうかを選択するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで [Dashboards] (ダッシュボード) を選択し、ダッシュボードをクリックします。
3. ウィジェットを選択し、[アクション]、[編集] の順に選択します。
4. [グラフのオプション] タブを選択します。
5. [Live Data] の下のチェックボックスをオンまたはオフにします。

アニメーション化されたダッシュボードの表示

時間の経過とともにキャプチャされた CloudWatch メトリクスデータを再生する、アニメーション化されたダッシュボードを表示できます。これにより、傾向の把握、プレゼンテーションの作成、問題の発生後の分析に役立ちます。

ダッシュボードのアニメーション化されたウィジェットには、ラインウィジェット、積み上げ面ウィジェット、数値ウィジェット、メトリクスエクスプローラーウィジェットが含まれます。円グラフ、棒グラフ、テキストウィジェット、ログウィジェットはダッシュボードに表示されますが、アニメーション化されません。

アニメーション化されたダッシュボードを表示するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで、ダッシュボードを選択します。
3. ダッシュボード名を選択します。
4. [アクション]、[ダッシュボードの再生] の順に選択します
5. (オプション) 既定では、アニメーションを開始すると、アニメーションはスライディングウィンドウとして表示されます。アニメーションをポイントごとのアニメーションとして表示する場合は、アニメーションを一時停止している間に虫眼鏡アイコンを選択し、ズームをリセットします。
6. アニメーションを開始するには、[再生] ボタンを選択します。[戻る] ボタンと [進む] ボタンを選択して、他の時点に移動することもできます。
7. (オプション) アニメーションのタイムウィンドウを変更するには、カレンダーを選択し、期間を選択します。
8. アニメーションの速度を変更するには、[自動速度] を選択し、新しい速度を選択します。
9. 終了したら、[アニメーションの終了] を選択します。

CloudWatch ダッシュボードをお気に入りリストに追加する

CloudWatch コンソールでは、ダッシュボード、アラーム、ロググループをお気に入りリストに追加できます。ナビゲーションペインからお気に入りリストにアクセスできます。次の手順では、ダッシュボードをお気に入りリストに追加する方法について説明します。

ダッシュボードをお気に入りリストに追加するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで、ダッシュボードを選択します。
3. ダッシュボードのリストで、お気に入りに追加するダッシュボード名の横にある星記号を選択します。

   - (オプション) リストからダッシュボードを選択し、ダッシュボード名の横にある星記号を選択して、ダッシュボードをお気に入りに追加することもできます。
4. お気に入りリストにアクセスするには、ナビゲーションペインで [Favorites and recents] (お気に入りと最近のアクセス) を選択します。メニューには 2 つの列があります。1 つの列にはお気に入りのダッシュボード、アラーム、ロググループが含まれ、もう 1 つの列には最近アクセスしたダッシュボード、アラーム、ロググループが含まれています。

> **Tip**
>
> ダッシュボード、アラーム、ロググループは、CloudWatch コンソールのナビゲーションペインにある [Favorites and recents] (お気に入りと最近のアクセス) メニューから、お気に入りに追加できます。[Recently visited] (最近のアクセス) 列で、お気に入りに追加するダッシュボードにカーソルを合わせ、その横にある星記号をクリックします。

CloudWatch ダッシュボードの期間の上書き設定または更新間隔を変更する

このダッシュボードに追加されたグラフの期間設定を保持または変更する方法を指定できます。

自動の期間または保持される時間範囲がウィジェットに適用されると、グラフの全体的な時間範囲が設定した期間に影響する可能性があります。

- 時間範囲が 1 日以内の場合、期間の設定は変更されません。
- 時間範囲が 1 日から 3 日の場合、5 分以内に設定された期間は 5 分に変更されます。
- 時間範囲が 3 日を超える場合、1 時間以内に設定された期間は 1 時間に変更されます。

以降のステップでは、コンソールを使用して期間の上書きオプションを変更する方法について説明します。このほか、ダッシュボードの JSON 構造にある `period0verride` フィールドを使用して変

更することもできます。詳細については、「Dashboard Body Overall Structure」を参照してください。

期間の上書きオプションを変更するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. [Actions] を選択します。
3. [Period override] で、次のいずれかを選択します。

   - [Auto] を選択すると、各グラフのメトリクスの期間を、ダッシュボードの期間に自動的に合わせることができます。
   - [Do not override] を選択すると、各グラフの期間設定が常に保持されます。
   - 他のオプションの 1 つを選択して、ダッシュボードに追加されたグラフで、常に選択された時間が期間設定として適用されるようにします。

   ダッシュボードが閉じられるか、ブラウザが更新されると、[Period override (期間の上書き)] は常に [自動] に戻ります。[Period override (期間の上書き)] の異なる設定は保存できません。

CloudWatch ダッシュボードのデータの更新頻度を変更できます。

ダッシュボードの更新間隔を変更する方法

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで [Dashboards] (ダッシュボード) を選択し、ダッシュボードをクリックします。
3. [更新オプション] メニュー (右上) で、[10 秒]、[1 分]、[2 分]、[5 分]、または [15 分] を選択します。

CloudWatch ダッシュボードの時間範囲またはタイムゾーン形式を変更する

ダッシュボードのデータを表示する時間範囲を分、時間、日、週単位に変更できます。タイムゾーン形式を変更して、ダッシュボードのデータを UTC またはローカルタイムで表示することもできます。ローカルタイムは、コンピュータのオペレーティングシステムで指定されているタイムゾーンです。

> **Note**
>
> 高解像度メトリクス数が 100 以上含まれているグラフでダッシュボードを作成する場合は、時間範囲を 1 時間以内に設定することをお勧めします。詳細については、「高解像度のメトリクス」を参照してください。

> **Note**
>
> ダッシュボードの時間範囲がダッシュボード上のウィジェットに使用されている期間よりも短い場合、次が実行されます。
>
> - ダッシュボードの時間範囲よりも長い場合でも、そのウィジェットの 1 つの期間に対応するデータ量が表示されるようにウィジェットが変更されます。これにより、グラフ上に少なくとも 1 つのデータポイントが表示されるようになります。
> - このデータポイントの期間の開始時刻は、少なくとも 1 つのデータポイントが表示されるように逆算して調整されます。

New console

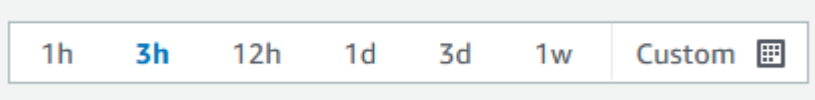
ダッシュボードで時間範囲を変更する方法

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで [Dashboards] (ダッシュボード) を選択し、ダッシュボードをクリックします。
3. ダッシュボード画面から、次のいずれかを実行します。
   - ダッシュボードの上部で、いずれかの事前定義済みの時間範囲を選択します。これらの期間は、1 時間から 1 週間です ([1h] (1 時間)、[3h] (3 時間)、[12h] (12 時間)、[1d] (1 日)、[1w] (1 週間))。
   - または、次のカスタム時間範囲のオプションのいずれかを選択できます。
     - [Custom] (カスタム) を選択してから、[Relative] (相対値) を選択します。1 分から 15 か月の期間を選択します。
     - [Custom] (カスタム) メニューを選択してから、[Absolute] (絶対値) タブを選択します。カレンダーまたはテキストフィールドを使用して、時間範囲を指定します。

> **Tip**
>
> グラフの時間範囲を変更する場合、集約期間が [Auto] (自動) に設定されていると、CloudWatch により期間が変更される可能性があります。期間を手動で設定するには、[Actions] (アクション) ドロップダウンを選択してから、[Period override] (期間の上書き) を選択します。

ダッシュボードのタイムゾーン形式を変更するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで [Dashboards] (ダッシュボード) を選択し、ダッシュボードをクリックします。
3. ダッシュボードの上部で、[カスタム] を選択します。



4. 表示されたボックスの右上隅で、ドロップダウンから [UTC] または [Local time] (現地時間) を選選択します。
5. [Apply] を選択します。

Old console

ダッシュボードで時間範囲を変更する方法

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで [Dashboards] (ダッシュボード) を選択し、ダッシュボードをクリックします。
3. ダッシュボード画面から、次のいずれかを実行します。
   - ダッシュボードの上部で、いずれかの事前定義済みの時間範囲を選択します。これらの期間は、1 時間から 1 週間です ([1h] (1 時間)、[3h] (3 時間)、[12h] (12 時間)、[1d] (1 日)、[3d] (3 日)、[1w] (1 週間))。
   - または、次のカスタム時間範囲のオプションのいずれかを選択できます。
     - [Custom] (カスタム) ドロップダウンを選択してから、[Relative] (相対値) タブを選択します。1 分 ~ 15 か月の定義済み範囲から 1 つを選択します。

- [Custom] (カスタム) ドロップダウンを選択してから、[Absolute] (絶対値) タブを選択します。カレンダーまたはテキストフィールドを使用して、時間範囲を指定します。

> **Tip**
>
> グラフの時間範囲を変更する場合、集約期間が [Auto] (自動) に設定されていると、CloudWatch により期間が変更される可能性があります。期間を手動で設定するには、[Actions] (アクション) ドロップダウンを選択してから、[Period override] (期間の上書き) を選択します。

ダッシュボードのタイムゾーン形式を変更するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで [Dashboards] (ダッシュボード) を選択し、ダッシュボードをクリックします。
3. ダッシュボード画面の右上隅で、ドロップダウンから [Custom] (カスタム) を選択します。
4. 表示されたボックスの右上隅で、ドロップダウンから [UTC] または [Local timezone] (ローカルタイムゾーン) を選択します。

Amazon CloudWatch メトリクスを使用する

メトリクスとは、システムのパフォーマンスに関するデータです。デフォルトでは、多くのサービスはリソース (Amazon EC2 インスタンス、Amazon EBS ボリューム、Amazon RDS DB インスタンスなど) に対して無料のメトリクスを提供しています。また、Amazon EC2 インスタンスなど一部のリソースの詳細モニターリングを有効にしたり、独自のアプリケーションメトリクスを発行したりできます。Amazon CloudWatch は、検索、グラフ表示、アラームに備えて、アカウント内のすべてのメトリクス (AWS リソースメトリクスと、お使いのアプリケーションメトリクスの両方) をロードすることができます。

メトリクスデータは 15 か月間保持されるため、最新データと履歴データの両方が表示できます。

コンソールでメトリクスをグラフ表示する際には、CloudWatch Metrics Insights を使用できます。CloudWatch Metrics Insights は、すべてのメトリクスに関するトレンドとパターンをリアルタイムで識別するための、高パフォーマンスの SQL クエリエンジンです。

内容

基本モニターリングと詳細モニターリング

CloudWatch には、基本モニターリングと詳細モニターリングの 2 つのカテゴリのモニターリングが用意されています。

多くの AWS のサービスでは、顧客には無料で、デフォルトのメトリクスのセットを CloudWatch に公開することで、基本的なモニターリングを提供しています。デフォルトでは、これら AWS のサービス のいずれかを使用し始めると、基本モニターリングが自動的に有効になります。基本モニターリングを提供するサービスのリストについては、「CloudWatch メトリクスを発行する AWS のサービス」を参照してください。

詳細モニターリングは、一部のサービスでのみ提供されます。料金も発生します。AWS のサービスで使用するには、アクティブにするように選択する必要があります。料金の詳細については、「Amazon CloudWatch の料金」を参照してください。

詳細モニターリングオプションは、提供するサービスによって異なります。例えば、Amazon EC2 詳細モニターリングでは、Amazon EC2 の基本モニターリングで使用される 5 分間隔ではなく、高い頻度の 1 分間隔で公開されるメトリクスが用意されています。Amazon S3 および Amazon Managed Streaming for Apache Kafka の詳細モニターリングは、よりきめ細かなメトリクスになっています。

異なる AWS のサービスでは、詳細モニターリングの名前も異なります。例えば、Amazon EC2 では、詳細モニターリングと呼ばれ、AWS Elastic Beanstalk では、拡張モニターリングと呼ばれ、Amazon S3 ではリクエストメトリクスと呼ばれます。

Amazon EC2 の詳細なモニターリングを使用すると、Amazon EC2 リソースをより適切に管理できるため、傾向を見つけてアクションを迅速に行うことができます。Amazon S3 リクエストメトリクスは、1 分間隔で利用可能で、オペレーションの問題をすばやく特定して対応するのに役立ちます。Amazon MSK で、`PER_BROKER`、`PER_TOPIC_PER_BROKER`、または `PER_TOPIC_PER_PARTITION` レベルのモニターリングを有効にすると、追加のメトリクスを取得して可視性が高まります。

次の表に、詳細モニターリングを提供するサービスの一覧を示します。また、詳細モニターリングを詳細に説明し、アクティブ化する方法について説明するこれらのサービスのドキュメントへのリンクも含まれています。

サービス	ドキュメント
Amazon API Gateway	API Gateway メトリクスのディメンション
Amazon CloudFront	CloudFront ディストリビューシ

サービス	ドキュメント
	ョンの追加のメトリクスの表示
Amazon EC2	インスタンスの詳細モニターリングを有効または無効にする
Elastic Beanstalk	拡張ヘルスレポートおよびモニターリング
Amazon Kinesis Data Streams	拡張シャードレベルメトリクス
Amazon MSK	CloudWatch でモニターリングするための Amazon MSK メトリクス
Amazon S3	CloudWatch の Amazon S3 リクエストメトリクス
Amazon SES	Amazon SES イベント発行を使用して CloudWatch の詳細モニタリングメトリクスを収集します。

また、次の表に示すように、CloudWatch ではより詳細なメトリクスと事前作成されたダッシュボードを備えた、一部の AWS のサービス用のすぐに使えるモニタリングソリューションを提供しています。

サービス	機能に関するドキュメント
Lambda	Lambda Insights
Amazon ECS	Container Insights for Amazon ECS
Amazon EKS	Container Insights for Amazon EKS and Kubernetes

CloudWatch Metrics Insights を使用してメトリクスをクエリする

CloudWatch Metrics Insights は、高性能な SQL クエリエンジンであり、これにより、メトリクスに対し大量のクエリを実行できます。CloudWatch メトリクスの全体について、トレンドとパターンをリアルタイムで識別できます。

また、単一の時系列を返すどの Metrics Insights クエリにでも、アラームを設定できます。これは、インフラストラクチャやアプリケーションのフリートで集約されたメトリクスを監視するアラームを作成する場合に特に役立ちます。アラームを一度作成すると、リソースがフリートに追加されたり、フリートから削除されたりするたびに動的に調整されます。

CloudWatch Metrics Insights クエリエディタを使用すると、コンソールで CloudWatch Metrics Insights クエリを実行できます。また、`GetMetricData` や `PutDashboard` を実行して、AWS CLI や AWS SDK で CloudWatch Metrics Insights クエリを実行することもできます。CloudWatch Metrics Insights クエリエディタでクエリを実行した場合には、料金がかかりません。CloudWatch の料金の詳細については、Amazon CloudWatch の料金をご覧ください。

CloudWatch Metrics Insights クエリエディタでは、事前構築済みのさまざまなサンプルクエリを選択できるほか、独自のクエリを作成することもできます。クエリを作成すると、ビルダービューを使用

して既存のメトリクスとディメンションを参照できます。あるいは、エディタビューを使用して、手動でクエリを記述できます。

このほか、自然言語を使用して CloudWatch Metrics Insights クエリを作成することもできます。そのためには、どのようなデータを探しているのかを質問したり、具体的に説明したりしてみてください。AI 支援機能が働いて、プロンプトに基づいてクエリが生成され、クエリの仕組みを説明した文が 1 行ずつ表示されます。詳細については、「自然言語を使用した CloudWatch Metrics Insights クエリの生成と更新」を参照してください。

Metrics Insights では、大量のクエリを実行できます。GROUP BY 句を使用すると、メトリクスを特定のディメンション値ごとに個別の時系列にリアルタイムでグループ化できます。Metrics Insights クエリには ORDER BY 機能があるため、Metrics Insights を使用して「上位 N」といったタイプのクエリを作成できます。例えば、「上位 N」タイプのクエリでは、アカウント内の何百万ものメトリクスをスキャンして、CPU の消費率が高い上位 10 個のインスタンスを返すことができます。アプリケーションのレイテンシーの問題を特定して修正する場合に効果的です。

トピック

- クエリを作成する
- Metrics Insights のクエリコンポーネントと構文
- Metrics Insights クエリでアラームを作成する
- メトリクス数式のクエリで Metrics Insights を使用する
- 自然言語を使用した CloudWatch Metrics Insights クエリの生成と更新
- SQL 推論
- Metrics Insights のサンプルクエリ
- Metrics Insights の制限
- Metrics Insights 用語集
- Metrics Insights のトラブルシューティング

クエリを作成する

CloudWatch Metrics Insights のクエリは、CloudWatch コンソール、AWS CLI、または AWS SDK から実行します。コンソールで実行されるクエリは無料です。CloudWatch の料金の詳細については、Amazon CloudWatch の料金をご覧ください。

AWS SDK による Metrics Insights クエリの実行に関する詳細は、「GetMetricData」でご確認ください。

CloudWatch コンソールを使用してクエリを実行するには、以下の手順に従います。

Metrics Insights を使用してメトリクスをクエリするには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで、[Metrics]、[All metrics] を選択します。
3. [Queries] (クエリ) タブを開きます。
4. (オプション) 事前構築のサンプルクエリを実行するには、[Add query] (クエリを追加) をクリックした上で、実行するクエリを選択します。選択したクエリの機能に不足がない場合は、この手順の残りをスキップできます。または、[Editor] (エディタ) をクリックし、サンプルクエリの編集を行った上で、[Run] (実行) をクリックしてそのクエリを実行できます。
5. 独自のクエリを作成する際は、[Builder] (ビルダー)ビューと [Editor] (エディタ) ビューのどちらか、あるいはその両方を組み合わせて使用します。この 2 つのビューはいつでも切り替えが可能で、どちらからも進行中の作業を確認できます。

   [Builder] (ビルダー) ビューでは、メトリクスの名前空間、メトリクス名、フィルタ、グループ、および順序付けのオプションを閲覧し選択できます。これらの各オプションについて、それぞれを環境から選択するためのリストが、クエリビルダーから提供されます。

   [Editor] (エディタ) では、自分のためのクエリを記述していくことができます。このエディタでは、その時点までにタイプした文字に基づいた入力の候補が表示されます。
6. クエリの内容に問題がなければ、[Run] (実行) をクリックします。
7. (オプション) グラフ表示したクエリを編集するには、[Graphed metrics] (グラフ化したメトリクス) タブを開き、[Details] (詳細) 列でクエリ式の横にある編集アイコンクリックするという方法もあります。
8. (オプション) グラフからクエリを削除する場合は、[Graphed metrics] (グラフ化したメトリクス) をクリックし、クエリが表示されている行の右側にある [X] アイコンをクリックします。

Metrics Insights のクエリコンポーネントと構文

CloudWatch Metrics Insights の構文は以下のとおりです。

```
SELECT FUNCTION(metricName)
FROM namespace | SCHEMA(...)
[ WHERE labelKey OPERATOR labelValue [AND ... ] ]
[ GROUP BY labelKey [ , ... ] ]
[ ORDER BY FUNCTION() [ DESC | ASC ] ]
```

```
[ LIMIT number ]
```

Metrics Insights のクエリで使用可能な句は以下のとおりです。キーワードでは大文字と小文字は区別されませんが、メトリクスの名前、名前空間、ディメンションなど、メトリクスの識別子については大文字と小文字が区別されます。

SELECT

必須。各タイムバケットの (指定された期間によって決定される) 観測値を集計するための関数を指定します。同時に、クエリするメトリクスの名前も指定します。

FUNCTION での有効な値は、AVG、COUNT、MAX、MIN、および SUM です。

- AVG では、クエリで一致があった観測値の平均を計算します。
- COUNT では、クエリで一致した観測値のカウントを返します。
- MAX では、クエリで一致があった観測値の最大値を返します。
- MIN では、クエリで一致があった観測値の最小値を返します。
- SUM では、クエリで一致があった観測値を合計し、その値を返します。

FROM

必須。メトリクスのソースを指定します。クエリされるメトリクスを含むメトリクス名前空間、あるいは、SCHEMA テーブル関数のどちらかを指定します。メトリクス名前空間の例としては、"AWS/EC2"、"AWS/Lambda" などを初めとして、カスタムメトリクス用にユーザーが作成したメトリクス名前空間なども含まれます。

/ を含む (または、文字、数字、アンダースコア以外の文字を含む) メトリクス名前空間は、二重引用符で囲む必要があります。詳細については、「引用符や エスケープ文字を使用すべき場合を教えてください。」を参照してください。

SCHEMA

FROM 句内で使用できるオプションのテーブル関数。SCHEMA は、クエリ結果をリストされたディメンションと完全に一致したメトリクスからのもの、あるいは、ディメンションを持たないメトリクスからのものに絞り込みます。

SCHEMA 句を使用する際は、クエリ対象のメトリクス名前空間を指定するために、(最初の引数として) 少なくとも 1 つを渡す必要があります。SCHEMA において、この名前空間引数のみを指定した場合は、クエリ結果の範囲がディメンションを持たないメトリクスのみに絞られます。

SCHEMA で、名前空間引数の後に他の引数を指定する場合、それらの追加の引数はラベルキーにする必要があります。ラベルキーでは、ディメンション名を指定します。このラベルキーを 1 つ以上指定することで、ディメンションセットが正確に一致するメトリクスのみに、クエリ結果の範囲が絞り込まれます。これらのラベルキーの順序は任意です。

例:

- SELECT AVG(CPUUtilization) FROM "AWS/EC2" は、AWS/EC2 名前空間内にあるすべての CPUUtilization メトリクスと一致し、ディメンションは関係ありません。結果には、単一の集計時系列が返されます。
- SELECT AVG(CPUUtilization) FROM SCHEMA("AWS/EC2") では、AWS/EC2 名前空間内でディメンションが定義されていない CPUUtilization メトリクスのみが一致します。
- SELECT AVG(CPUUtilization) FROM SCHEMA("AWS/EC2", InstanceId) では、厳密に 1 つのディメンション (InstanceId) で CloudWatch に報告された CPUUtilization メトリクスのみと一致します。
- SELECT SUM(RequestCount) FROM SCHEMA("AWS/ApplicationELB", LoadBalancer, AvailabilityZone) では、厳密に 2 つのディメンション (LoadBalancer と AvailabilityZone) で AWS/ApplicationELB から CloudWatch に報告された RequestCount メトリクスのみと一致します。

WHERE

オプション。1 つ以上のラベルキーで特定のラベル値を使用しながら指定した式と一致するメトリクスのみに、結果をフィルタリングします。例えば、WHERE InstanceType = 'c3.4xlarge' は、結果を c3.4xlarge インスタンスタイプのみにフィルタリングします。また、WHERE InstanceType != 'c3.4xlarge'では、c3.4xlarge を除くすべてのインスタンスタイプに結果をフィルタリングします。

モニタリングアカウントでクエリを実行する場合、WHERE AWS.AccountId を使用して、指定したアカウントのみに結果を制限できます。例えば、WHERE AWS.AccountId=444455556666 は 444455556666 アカウントのみのメトリクスをクエリします。クエリをモニタリングアカウント自体のメトリクスにのみ制限するには、WHERE AWS.AccountId=CURRENT_ACCOUNT_ID() を使用します。

ラベル値は、常に一重引用符で囲む必要があります。

サポートされている演算子

WHERE 句では以下の演算子がサポートされます。

- = ラベル値は指定された文字列と一致することを表します。
- !=ラベル値は、指定した文字列と一致しないことを表します。
- AND は、指定された 2 つの条件がともに true である場合に一致することを表します。AND キーワードは、2 つ以上の条件を指定するために使用することもできます。

GROUP BY

オプション。クエリ結果を複数の時系列にグループ化します。各時系列は、指定したラベルキーもしくは他のキーでの、さまざまな値に対応します。例えば `GROUP BY InstanceId` を使用すると、`InstanceId` の各値に応じて異なる時系列が返されます。`GROUP BY ServiceName, Operation` を使用すると、`ServiceName` と `Operation` の値による可能な組み合わせごとに、それぞれ時系列が作成されます。

GROUP BY 句では、デフォルトで、結果がアルファベットの昇順に並べられます。その際は、この GROUP BY 句で指定されたラベルシーケンスが使用されます。この結果の順序を変更するには、クエリに ORDER BY 句を追加します。

モニタリングアカウントでクエリを実行する場合、`GROUP BY AWS.AccountId` を使用して、各結果をその生成元のアカウントに基づいてグループ化できます。

> **Note**
>
> 一致するメトリクスの一部に、GROUP BY 句で指定された特定のラベルキーを含まないものがある場合、`Other` という名前の NULL グループが返されます。例えば、`GROUP BY ServiceName, Operation` を指定している場合で、返されたメトリクスの一部にディメンションとして `ServiceName` を含まないものがあると、これらのメトリクスは、`Other` という値の `ServiceName` を持つものとして表示されます。

ORDER BY

オプション。クエリが複数の時系列を返す場合に、それら返される時系列で使用する順序を指定します。この順序は、[ORDER BY] 句で指定した [FUNCTION] が検出した値に基づいて決定されます。FUNCTION は、返されたそれぞれの時系列から単一のスカラー値を算出し、この値が、順序を決定するために使用されます。

また順序には、ASC (昇順) か DESC (降順) のどちらを使用するかも指定します。この指定を省略した場合は、デフォルトで昇順の ASC が使用されます。

例えば ORDER BY MAX() DESC 句では、対象の時間範囲内での観測結果を、最大データポイントにより降順で順序付けします。つまり、最大データポイントが最も多い時系列が、最初の結果として返されます。

ORDER BY 句内で使用可能な関数は、AVG()、COUNT()、MAX()、MIN()、および SUM() です。

ORDER BY 句を LIMIT 句とともに使用した場合、返される結果は「Top N」クエリになります。個別のクエリが返すことができる時系列は 500 個以下であるため、ORDER BY は、クエリから多数のメトリクスを返させたい場合にも役立ちます。クエリで 500 個を超える時系列 が一致した場合に ORDER BY 句を使用していると、時系列が並べ替えられ、その順序内で最初の 500 個までの時系列を結果として返すことができます。

LIMIT

オプション。クエリによって返される時系列の数を、指定した値に制限します。指定できる最大値は 500 です。また、クエリに LIMIT を指定しない場合でも、返すことができる時系列は 500 個以下です。

LIMIT 句を ORDER BY 句とともに使用すると、「Top N」のクエリが返されます。

引用符やエスケープ文字を使用すべき場合を教えてください。

クエリで指定するラベル値は、常に一重引用符で囲む必要があります。例えば、SELECT MAX(CPUUtilization) FROM "AWS/EC2" WHERE AutoScalingGroupName = 'my-production-fleet' のようにします。

文字、数字、アンダースコア (_) 以外の文字を含むメトリクスの名前空間、メトリクス名、ラベルキーは、二重引用符で囲む必要があります。例としては、SELECT MAX("My.Metric") のようになります。

これらのいずれかに二重引用符または一重引用符自体が含まれている (例: Bytes"Input") 場合は、SELECT AVG("Bytes\"Input\"") のようにして、各引用符をバックスラッシュでエスケープする必要があります。

メトリクス名前空間、メトリクス名、またはラベルキーに、Metrics Insights での予約キーワードが含まれている場合は、これらも二重引用符で囲む必要があります。例えば、LIMIT という名前のメトリクスを指定するのであれば、SELECT AVG("LIMIT") のようになります。また、予約キーワードを含まない場合に、名前空間、メトリクス名、またはラベルを二重引用符で囲んだとしても、エラーとはなりません。

全予約キーワードの一覧については、「予約済みキーワード」を参照してください。

リッチクエリの作成手順

このセクションでは、利用可能なすべての句を使用してクエリを作成する際の完全な例を、ステップバイステップで解説します。

最初の例は次のクエリです。このクエリは、LoadBalancer と AvailabilityZone の両方のディメンションで収集された、Application Load Balancer のすべての RequestCount メトリクスを集計します。

```
SELECT SUM(RequestCount)
FROM SCHEMA("AWS/ApplicationELB", LoadBalancer, AvailabilityZone)
```

ここで、特定のロードバランサーからのメトリクスのみを表示したい場合は、WHERE 句を追加します。これにより、返されるメトリクスを LoadBalancer ディメンションの値が app/load-balancer-1 であるもののみに制限できます。

```
SELECT SUM(RequestCount)
FROM SCHEMA("AWS/ApplicationELB", LoadBalancer, AvailabilityZone)
WHERE LoadBalancer = 'app/load-balancer-1'
```

前述のクエリでは、このロードバランサーに関するすべてのアベイラビリティーゾーンからの RequestCount メトリクスが、1 つの時系列内に集計されます。アベイラビリティーゾーンごとに異なる時系列を表示したい場合は、GROUP BY 句を追加します。

```
SELECT SUM(RequestCount)
FROM SCHEMA("AWS/ApplicationELB", LoadBalancer, AvailabilityZone)
WHERE LoadBalancer = 'app/load-balancer-1'
GROUP BY AvailabilityZone
```

次に、これらの結果を並べ替えて、最高値を最初に表示する場合を考えます。以下の ORDER BY 句では、クエリの一定時間範囲内にある各時系列によって報告された最大値について、対応する時系列が降順で順序付けられます。

```
SELECT SUM(RequestCount)
FROM SCHEMA("AWS/ApplicationELB", LoadBalancer, AvailabilityZone)
WHERE LoadBalancer = 'app/load-balancer-1'
GROUP BY AvailabilityZone
ORDER BY MAX() DESC
```

さらに、主として「Top N」タイプのクエリを表示したい場合には、LIMIT 句を使用します。この最後の例では、MAX 値が最上位の 5 つに入っている時系列のみに結果を制限しています。

```
SELECT SUM(RequestCount)
FROM SCHEMA("AWS/ApplicationELB", LoadBalancer, AvailabilityZone)
WHERE LoadBalancer = 'app/load-balancer-1'
GROUP BY AvailabilityZone
ORDER BY MAX() DESC
LIMIT 5
```

クロスアカウントクエリの例

こうした例は、CloudWatch クロスアカウントオブザーバビリティにモニタリングアカウントとして設定されたアカウントで実行する場合に有効です。

次の例では、ソースアカウント 123456789012 内の Amazon EC2 インスタンスをすべて検索して、その平均を返しています。

```
SELECT AVG(CpuUtilization)
FROM "AWS/EC2"
WHERE AWS.AccountId ='123456789012'
```

次の例では、リンクされたすべてのソースアカウント内の AWS/EC2 で CPUUtilization メトリクスをクエリし、アカウント ID とインスタンスタイプで結果をグループ化しています。

```
SELECT AVG(CpuUtilization)
FROM "AWS/EC2"
GROUP BY AWS.AccountId, InstanceType
```

次の例では、モニタリングアカウント自体で CPUUtilization をクエリしています。

```
SELECT AVG(CpuUtilization)
FROM "AWS/EC2"
WHERE AWS.AccountId = CURRENT_ACCOUNT_ID()
```

予約済みキーワード

以下が、CloudWatch Metrics Insights での予約キーワードです。これらの単語が、クエリ内の名前空間、メトリクス名、またはラベルキーに含まれている場合、その単語は二重引用符で囲む必要があります。予約キーワードでは大文字と小文字が区別されません。

```
"ABORT" "ABORTSESSION" "ABS" "ABSOLUTE" "ACCESS" "ACCESSIBLE" "ACCESS_LOCK" "ACCOUNT"
 "ACOS" "ACOSH" "ACTION" "ADD" "ADD_MONTHS"
"ADMIN" "AFTER" "AGGREGATE" "ALIAS" "ALL" "ALLOCATE" "ALLOW" "ALTER" "ALTERAND" "AMP"
 "ANALYSE" "ANALYZE" "AND" "ANSIDATE" "ANY" "ARE" "ARRAY",
"ARRAY_AGG" "ARRAY_EXISTS" "ARRAY_MAX_CARDINALITY" "AS" "ASC" "ASENSITIVE" "ASIN"
 "ASINH" "ASSERTION" "ASSOCIATE" "ASUTIME" "ASYMMETRIC" "AT",
"ATAN" "ATAN2" "ATANH" "ATOMIC" "AUDIT" "AUTHORIZATION" "AUX" "AUXILIARY" "AVE"
 "AVERAGE" "AVG" "BACKUP" "BEFORE" "BEGIN" "BEGIN_FRAME" "BEGIN_PARTITION",
"BETWEEN" "BIGINT" "BINARY" "BIT" "BLOB" "BOOLEAN" "BOTH" "BREADTH" "BREAK" "BROWSE"
 "BT" "BUFFERPOOL" "BULK" "BUT" "BY" "BYTE" "BYTEINT" "BYTES" "CALL",
"CALLED" "CAPTURE" "CARDINALITY" "CASCADE" "CASCADED" "CASE" "CASESPECIFIC" "CASE_N"
 "CAST" "CATALOG" "CCSID" "CD" "CEIL" "CEILING" "CHANGE" "CHAR",
"CHAR2HEXINT" "CHARACTER" "CHARACTERS" "CHARACTER_LENGTH" "CHARS" "CHAR_LENGTH" "CHECK"
 "CHECKPOINT" "CLASS" "CLASSIFIER" "CLOB" "CLONE" "CLOSE" "CLUSTER",
"CLUSTERED" "CM" "COALESCE" "COLLATE" "COLLATION" "COLLECT" "COLLECTION" "COLLID"
 "COLUMN" "COLUMN_VALUE" "COMMENT" "COMMIT" "COMPLETION" "COMPRESS" "COMPUTE",
"CONCAT" "CONCURRENTLY" "CONDITION" "CONNECT" "CONNECTION" "CONSTRAINT" "CONSTRAINTS"
 "CONSTRUCTOR" "CONTAINS" "CONTAINSTABLE" "CONTENT" "CONTINUE" "CONVERT",
"CONVERT_TABLE_HEADER" "COPY" "CORR" "CORRESPONDING" "COS" "COSH" "COUNT" "COVAR_POP"
 "COVAR_SAMP" "CREATE" "CROSS" "CS" "CSUM" "CT" "CUBE" "CUME_DIST",
"CURRENT" "CURRENT_CATALOG" "CURRENT_DATE" "CURRENT_DEFAULT_TRANSFORM_GROUP"
 "CURRENT_LC_CTYPE" "CURRENT_PATH" "CURRENT_ROLE" "CURRENT_ROW" "CURRENT_SCHEMA",
"CURRENT_SERVER" "CURRENT_TIME" "CURRENT_TIMESTAMP" "CURRENT_TIMEZONE"
 "CURRENT_TRANSFORM_GROUP_FOR_TYPE" "CURRENT_USER" "CURRVAL" "CURSOR" "CV" "CYCLE"
 "DATA",
"DATABASE" "DATABASES" "DATABLOCKSIZE" "DATE" "DATEFORM" "DAY" "DAYS" "DAY_HOUR"
 "DAY_MICROSECOND" "DAY_MINUTE" "DAY_SECOND" "DBCC" "DBINFO" "DEALLOCATE" "DEC",
"DECFLOAT" "DECIMAL" "DECLARE" "DEFAULT" "DEFERRABLE" "DEFERRED" "DEFINE" "DEGREES"
 "DEL" "DELAYED" "DELETE" "DENSE_RANK" "DENY" "DEPTH" "DEREF" "DESC" "DESCRIBE",
"DESCRIPTOR" "DESTROY" "DESTRUCTOR" "DETERMINISTIC" "DIAGNOSTIC" "DIAGNOSTICS"
 "DICTIONARY" "DISABLE" "DISABLED" "DISALLOW" "DISCONNECT" "DISK" "DISTINCT",
"DISTINCTROW" "DISTRIBUTED" "DIV" "DO" "DOCUMENT" "DOMAIN" "DOUBLE" "DROP" "DSSIZE"
 "DUAL" "DUMP" "DYNAMIC" "EACH" "ECHO" "EDITPROC" "ELEMENT" "ELSE" "ELSEIF",
"EMPTY" "ENABLED" "ENCLOSED" "ENCODING" "ENCRYPTION" "END" "END-EXEC" "ENDING"
 "END_FRAME" "END_PARTITION" "EQ" "EQUALS" "ERASE" "ERRLVL" "ERROR" "ERRORFILES",
"ERRORTABLES" "ESCAPE" "ESCAPED" "ET" "EVERY" "EXCEPT" "EXCEPTION" "EXCLUSIVE" "EXEC"
 "EXECUTE" "EXISTS" "EXIT" "EXP" "EXPLAIN" "EXTERNAL" "EXTRACT" "FALLBACK
"FALSE" "FASTEXPORT" "FENCED" "FETCH" "FIELDPROC" "FILE" "FILLFACTOR" "FILTER" "FINAL"
 "FIRST" "FIRST_VALUE" "FLOAT" "FLOAT4" "FLOAT8" "FLOOR"
"FOR" "FORCE" "FOREIGN" "FORMAT" "FOUND" "FRAME_ROW" "FREE" "FREESPACE" "FREETEXT"
 "FREETEXTTABLE" "FREEZE" "FROM" "FULL" "FULLTEXT" "FUNCTION"
"FUSION" "GE" "GENERAL" "GENERATED" "GET" "GIVE" "GLOBAL" "GO" "GOTO" "GRANT" "GRAPHIC"
 "GROUP" "GROUPING" "GROUPS" "GT" "HANDLER" "HASH"
```

```
"HASHAMP" "HASHBAKAMP" "HASHBUCKET" "HASHROW" "HAVING" "HELP" "HIGH_PRIORITY" "HOLD"
 "HOLDLOCK" "HOUR" "HOURS" "HOUR_MICROSECOND" "HOUR_MINUTE"
"HOUR_SECOND" "IDENTIFIED" "IDENTITY" "IDENTITYCOL" "IDENTITY_INSERT" "IF" "IGNORE"
 "ILIKE" "IMMEDIATE" "IN" "INCLUSIVE" "INCONSISTENT" "INCREMENT"
"INDEX" "INDICATOR" "INFILE" "INHERIT" "INITIAL" "INITIALIZE" "INITIALLY" "INITIATE"
 "INNER" "INOUT" "INPUT" "INS" "INSENSITIVE" "INSERT" "INSTEAD"
"INT" "INT1" "INT2" "INT3" "INT4" "INT8" "INTEGER" "INTEGERDATE" "INTERSECT"
 "INTERSECTION" "INTERVAL" "INTO" "IO_AFTER_GTIDS" "IO_BEFORE_GTIDS"
"IS" "ISNULL" "ISOBID" "ISOLATION" "ITERATE" "JAR" "JOIN" "JOURNAL" "JSON_ARRAY"
 "JSON_ARRAYAGG" "JSON_EXISTS" "JSON_OBJECT" "JSON_OBJECTAGG"
"JSON_QUERY" "JSON_TABLE" "JSON_TABLE_PRIMITIVE" "JSON_VALUE" "KEEP" "KEY" "KEYS"
 "KILL" "KURTOSIS" "LABEL" "LAG" "LANGUAGE" "LARGE" "LAST"
"LAST_VALUE" "LATERAL" "LC_CTYPE" "LE" "LEAD" "LEADING" "LEAVE" "LEFT" "LESS" "LEVEL"
 "LIKE" "LIKE_REGEX" "LIMIT" "LINEAR" "LINENO" "LINES"
"LISTAGG" "LN" "LOAD" "LOADING" "LOCAL" "LOCALE" "LOCALTIME" "LOCALTIMESTAMP" "LOCATOR"
 "LOCATORS" "LOCK" "LOCKING" "LOCKMAX" "LOCKSIZE" "LOG"
"LOG10" "LOGGING" "LOGON" "LONG" "LONGBLOB" "LONGTEXT" "LOOP" "LOWER" "LOW_PRIORITY"
 "LT" "MACRO" "MAINTAINED" "MAP" "MASTER_BIND"
"MASTER_SSL_VERIFY_SERVER_CERT" "MATCH" "MATCHES" "MATCH_NUMBER" "MATCH_RECOGNIZE"
 "MATERIALIZED" "MAVG" "MAX" "MAXEXTENTS" "MAXIMUM" "MAXVALUE"
"MCHARACTERS" "MDIFF" "MEDIUMBLOB" "MEDIUMINT" "MEDIUMTEXT" "MEMBER" "MERGE" "METHOD"
 "MICROSECOND" "MICROSECONDS" "MIDDLEINT" "MIN" "MINDEX"
"MINIMUM" "MINUS" "MINUTE" "MINUTES" "MINUTE_MICROSECOND" "MINUTE_SECOND" "MLINREG"
 "MLOAD" "MLSLABEL" "MOD" "MODE" "MODIFIES" "MODIFY"
"MODULE" "MONITOR" "MONRESOURCE" "MONSESSION" "MONTH" "MONTHS" "MSUBSTR" "MSUM"
 "MULTISET" "NAMED" "NAMES" "NATIONAL" "NATURAL" "NCHAR" "NCLOB"
"NE" "NESTED_TABLE_ID" "NEW" "NEW_TABLE" "NEXT" "NEXTVAL" "NO" "NOAUDIT" "NOCHECK"
 "NOCOMPRESS" "NONCLUSTERED" "NONE" "NORMALIZE" "NOT" "NOTNULL"
"NOWAIT" "NO_WRITE_TO_BINLOG" "NTH_VALUE" "NTILE" "NULL" "NULLIF" "NULLIFZERO" "NULLS"
 "NUMBER" "NUMERIC" "NUMPARTS" "OBID" "OBJECT" "OBJECTS"
"OCCURRENCES_REGEX" "OCTET_LENGTH" "OF" "OFF" "OFFLINE" "OFFSET" "OFFSETS" "OLD"
 "OLD_TABLE" "OMIT" "ON" "ONE" "ONLINE" "ONLY" "OPEN" "OPENDATASOURCE"
"OPENQUERY" "OPENROWSET" "OPENXML" "OPERATION" "OPTIMIZATION" "OPTIMIZE"
 "OPTIMIZER_COSTS" "OPTION" "OPTIONALLY" "OR" "ORDER" "ORDINALITY" "ORGANIZATION"
"OUT" "OUTER" "OUTFILE" "OUTPUT" "OVER" "OVERLAPS" "OVERLAY" "OVERRIDE" "PACKAGE" "PAD"
 "PADDED" "PARAMETER" "PARAMETERS" "PART" "PARTIAL" "PARTITION"
"PARTITIONED" "PARTITIONING" "PASSWORD" "PATH" "PATTERN" "PCTFREE" "PER" "PERCENT"
 "PERCENTILE" "PERCENTILE_CONT" "PERCENTILE_DISC" "PERCENT_RANK" "PERIOD" "PERM"
"PERMANENT" "PIECESIZE" "PIVOT" "PLACING" "PLAN" "PORTION" "POSITION" "POSITION_REGEX"
 "POSTFIX" "POWER" "PRECEDES" "PRECISION" "PREFIX" "PREORDER"
"PREPARE" "PRESERVE" "PREVVAL" "PRIMARY" "PRINT" "PRIOR" "PRIQTY" "PRIVATE"
 "PRIVILEGES" "PROC" "PROCEDURE" "PROFILE" "PROGRAM" "PROPORTIONAL"
"PROTECTION" "PSID" "PTF" "PUBLIC" "PURGE" "QUALIFIED" "QUALIFY" "QUANTILE" "QUERY"
 "QUERYNO" "RADIANS" "RAISERROR" "RANDOM" "RANGE" "RANGE_N" "RANK"
```

```
"RAW" "READ" "READS" "READTEXT" "READ_WRITE" "REAL" "RECONFIGURE" "RECURSIVE" "REF"
 "REFERENCES" "REFERENCING" "REFRESH" "REGEXP" "REGR_AVGX" "REGR_AVGY"
"REGR_COUNT" "REGR_INTERCEPT" "REGR_R2" "REGR_SLOPE" "REGR_SXX" "REGR_SXY" "REGR_SYY"
 "RELATIVE" "RELEASE" "RENAME" "REPEAT" "REPLACE" "REPLICATION"
"REPOVERRIDE" "REQUEST" "REQUIRE" "RESIGNAL" "RESOURCE" "RESTART" "RESTORE" "RESTRICT"
 "RESULT" "RESULT_SET_LOCATOR" "RESUME" "RET" "RETRIEVE" "RETURN"
"RETURNING" "RETURNS" "REVALIDATE" "REVERT" "REVOKE" "RIGHT" "RIGHTS" "RLIKE" "ROLE"
 "ROLLBACK" "ROLLFORWARD" "ROLLUP" "ROUND_CEILING" "ROUND_DOWN"
"ROUND_FLOOR" "ROUND_HALF_DOWN" "ROUND_HALF_EVEN" "ROUND_HALF_UP" "ROUND_UP" "ROUTINE"
 "ROW" "ROWCOUNT" "ROWGUIDCOL" "ROWID" "ROWNUM" "ROWS" "ROWSET"
"ROW_NUMBER" "RULE" "RUN" "RUNNING" "SAMPLE" "SAMPLEID" "SAVE" "SAVEPOINT" "SCHEMA"
 "SCHEMAS" "SCOPE" "SCRATCHPAD" "SCROLL" "SEARCH" "SECOND" "SECONDS"
"SECOND_MICROSECOND" "SECQTY" "SECTION" "SECURITY" "SECURITYAUDIT" "SEEK" "SEL"
 "SELECT" "SEMANTICKEYPHRASETABLE" "SEMANTICSIMILARITYDETAILSTABLE"
"SEMANTICSIMILARITYTABLE" "SENSITIVE" "SEPARATOR" "SEQUENCE" "SESSION" "SESSION_USER"
 "SET" "SETRESRATE" "SETS" "SETSESSRATE" "SETUSER" "SHARE" "SHOW"
"SHUTDOWN" "SIGNAL" "SIMILAR" "SIMPLE" "SIN" "SINH" "SIZE" "SKEW" "SKIP" "SMALLINT"
 "SOME" "SOUNDEX" "SOURCE" "SPACE" "SPATIAL" "SPECIFIC" "SPECIFICTYPE"
"SPOOL" "SQL" "SQLEXCEPTION" "SQLSTATE" "SQLTEXT" "SQLWARNING" "SQL_BIG_RESULT"
 "SQL_CALC_FOUND_ROWS" "SQL_SMALL_RESULT" "SQRT" "SS" "SSL" "STANDARD"
"START" "STARTING" "STARTUP" "STAT" "STATE" "STATEMENT" "STATIC" "STATISTICS" "STAY"
 "STDDEV_POP" "STDDEV_SAMP" "STEPINFO" "STOGROUP" "STORED" "STORES"
"STRAIGHT_JOIN" "STRING_CS" "STRUCTURE" "STYLE" "SUBMULTISET" "SUBSCRIBER" "SUBSET"
 "SUBSTR" "SUBSTRING" "SUBSTRING_REGEX" "SUCCEEDS" "SUCCESSFUL"
"SUM" "SUMMARY" "SUSPEND" "SYMMETRIC" "SYNONYM" "SYSDATE" "SYSTEM" "SYSTEM_TIME"
 "SYSTEM_USER" "SYSTIMESTAMP" "TABLE" "TABLESAMPLE" "TABLESPACE" "TAN"
"TANH" "TBL_CS" "TEMPORARY" "TERMINATE" "TERMINATED" "TEXTSIZE" "THAN" "THEN"
 "THRESHOLD" "TIME" "TIMESTAMP" "TIMEZONE_HOUR" "TIMEZONE_MINUTE" "TINYBLOB"
"TINYINT" "TINYTEXT" "TITLE" "TO" "TOP" "TRACE" "TRAILING" "TRAN" "TRANSACTION"
 "TRANSLATE" "TRANSLATE_CHK" "TRANSLATE_REGEX" "TRANSLATION" "TREAT"
"TRIGGER" "TRIM" "TRIM_ARRAY" "TRUE" "TRUNCATE" "TRY_CONVERT" "TSEQUAL" "TYPE" "UC"
 "UESCAPE" "UID" "UNDEFINED" "UNDER" "UNDO" "UNION" "UNIQUE"
"UNKNOWN" "UNLOCK" "UNNEST" "UNPIVOT" "UNSIGNED" "UNTIL" "UPD" "UPDATE" "UPDATETEXT"
 "UPPER" "UPPERCASE" "USAGE" "USE" "USER" "USING" "UTC_DATE"
"UTC_TIME" "UTC_TIMESTAMP" "VALIDATE" "VALIDPROC" "VALUE" "VALUES" "VALUE_OF"
 "VARBINARY" "VARBYTE" "VARCHAR" "VARCHAR2" "VARCHARACTER" "VARGRAPHIC"
"VARIABLE" "VARIADIC" "VARIANT" "VARYING" "VAR_POP" "VAR_SAMP" "VCAT" "VERBOSE"
 "VERSIONING" "VIEW" "VIRTUAL" "VOLATILE" "VOLUMES" "WAIT" "WAITFOR"
"WHEN" "WHENEVER" "WHERE" "WHILE" "WIDTH_BUCKET" "WINDOW" "WITH" "WITHIN"
 "WITHIN_GROUP" "WITHOUT" "WLM" "WORK" "WRITE" "WRITETEXT" "XMLCAST" "XMLEXISTS"
"XMLNAMESPACES" "XOR" "YEAR" "YEARS" "YEAR_MONTH" "ZEROFILL" "ZEROIFNULL" "ZONE"
```

Metrics Insights クエリでアラームを作成する

Metrics Insights クエリでアラームを作成できます。これにより、後で更新しなくても複数のリソースを追跡するアラームを作成できます。クエリは、新しいリソースと変化したリソースをキャッチします。例えば、フリートの CPU 使用率を監視するアラームを作成し、アラームの作成後に起動した新しいインスタンスをアラームに自動的に評価させることができます。

CloudWatch クロスアカウントオブザーバビリティに対して設定されたモニタリングアカウントでは、Metrics Insights アラームを使用して、ソースアカウントとモニタリングアカウント自体内のリソースを監視できます。アラームクエリを特定のアカウントに制限する方法や、結果をアカウント ID でグループ化する方法の詳細については、「Metrics Insights のクエリコンポーネントと構文」の「`WHERE`」セクションと「`GROUP BY`」セクションを参照してください。

目次

Metrics Insights アラームを作成する

コンソールを使用して Metrics Insights クエリでアラームを作成するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで、[Metrics]、[All metrics] を選択します。
3. [Queries] (クエリ) タブを開きます。
4. (オプション) 事前構築のサンプルクエリを実行するには、[Add query] (クエリを追加) をクリックした上で、実行するクエリを選択します。または、[Editor] (エディタ) をクリックし、サンプルクエリの編集を行った上で、[Run] (実行) をクリックしてそのクエリを実行できます。
5. 独自のクエリを作成する際は、[Builder] ビューと [Editor] ビューのどちらか、あるいはその両方を組み合わせて使用します。この 2 つのビューはいつでも切り替えが可能で、どちらからも進行中の作業を確認できます。

   [Builder] (ビルダー) ビューでは、メトリクスの名前空間、メトリクス名、フィルタ、グループ、および順序付けのオプションを閲覧し選択できます。これらの各オプションについて、それぞれを環境から選択するためのリストが、クエリビルダーから提供されます。

   [Editor] (エディタ) では、自分のためのクエリを記述していくことができます。このエディタでは、その時点までにタイプした文字に基づいた入力の候補が表示されます。

> **Important**
>
> Metrics Insights クエリにアラームを設定するには、クエリが 1 つの時系列を返す必要があります。GROUP BY ステートメントが含まれる場合、GROUP BY ステートメントは、式の最終結果として 1 つの時系列のみを返す Metric Math 式の中にある必要があります。

6. クエリの内容に問題がなければ、[Run] (実行) をクリックします。
7. [アラームの作成] を選択します。
8. [Conditions (条件)] で、次のように指定します。

   a. [#####が次の時] で、メトリクスがしきい値より大きい、より小さい、またはしきい値と等しい必要があるかどうかを指定します。[than... (以下の値)] で、しきい値を指定します。

   b. [Additional configuration (追加設定)] を選択します。[Datapoints to alarm (アラームを発生させるデータポイント数)] で、アラームをトリガーするために ALARM 状態を維持する必要がある評価期間 (データポイント) の数を指定します。2 つの値が一致する場合は、該当する数の連続した期間でしきい値を超過したときに ALARM 状態に移行するアラームを作成します。

   N 個中 M 個のアラームを作成するには、2 番目の値よりも小さい数字を最初の値に指定します。詳細については、「アラームの評価」を参照してください。

   c. [Missing data treatment (欠落データの処理)] 、一部のデータポイントが欠落しているときのアラームによる対処方法を選択します。詳細については、「CloudWatch アラームの欠落データの処理の設定」を参照してください。
9. [Next (次へ)] を選択します。
10. [通知] で、アラームが ALARM 状態、OK 状態、または INSUFFICIENT_DATA 状態のときに通知するための SNS トピックを選択します。

    同じアラーム状態または複数の異なるアラーム状態について複数の通知を送信するには、[Add notification (通知の追加)] を選択します。

    アラームの通知を送信しない場合は、 [削除] を選択します。
11. アラームに伴って Auto Scaling、EC2、または Systems Manager アクションを実行するには、該当するボタンを選択し、アラーム状態と実行するアクションを選択します。アラームは、ALARM 状態になったときにのみ、Systems Manager のアクションを実行できま

す。Systems Manager のアクションの詳細については、「アラームから OpsItems を作成するように CloudWatch を設定する」および「Incident creation」を参照してください。

> **Note**
>
> SSM Incident Manager アクションを実行するアラームを作成するには、特定のアクセス許可が必要です。詳細については、AWS Systems Manager Incident Manager のアイデンティティベースのポリシーの例を参照してください。

12. 完了したら、[次へ] を選択します。
13. アラームの名前と説明を入力します。名前には ASCII 文字のみを使用します。続いて、[次へ] を選択します。
14. [Preview and create (プレビューして作成)] で、情報と条件が正しいことを確認し、[アラームの作成] を選択します。

AWS CLI を使用して Metrics Insights クエリでアラームを作成するには

- `put-metric-alarm` コマンドを使用して、`metrics` パラメータに Metrics Insights クエリを指定します。例えば、次のコマンドは、いずれかのインスタンスの CPU 使用率が 50% を超えた場合に ALARM 状態になるアラームを設定します。

```
aws cloudwatch put-metric-alarm --alarm-name Metrics-Insights-alarm --
evaluation-periods 1 --comparison-operator GreaterThanThreshold --metrics
 '[{"Id":"m1","Expression":"SELECT MAX(CPUUtilization) FROM SCHEMA(\"AWS/EC2\",
 InstanceId)", "Period":60}]' --threshold 50
```

部分的なデータのケース

アラームに使用された Metrics Insights クエリが 10,000 件を超えるメトリクスに一致する場合、アラームはクエリで見つかった最初の 10,000 件のメトリクスに基づいて評価されます。これは、アラームが部分的なデータに基づいて評価されていることを意味します。

次の方法を使用して、現在 Metrics Insights アラームが部分的なデータに基づいてアラームの状態を評価しているかどうかを確認できます。

- コンソールでアラームを選択して [Details] (詳細) ページを表示すると、そのページに「Evaluation warning: Not evaluating all data」(評価警告: すべてのデータを評価していません) というメッセージが表示されます。
- describe-alarms AWS CLI コマンドまたは DescribeAlarms API を使用すると、EvaluationState フィールドに PARTIAL_DATA という値が表示されます。

アラームは、部分的なデータの状態になったときにもイベントを Amazon EventBridge に公開するので、これらのイベントを監視する EventBridge ルールを作成できます。これらのイベントでは、evaluationState フィールドの値は PARTIAL_DATA です。次に例を示します。


```
{
    "version": "0",
    "id": "12345678-3bf9-6a09-dc46-12345EXAMPLE",
    "detail-type": "CloudWatch Alarm State Change",
    "source": "aws.cloudwatch",
    "account": "123456789012",
    "time": "2022-11-08T11:26:05Z",
    "region": "us-east-1",
    "resources": [
        "arn:aws:cloudwatch:us-east-1:123456789012:alarm:my-alarm-name"
    ],
    "detail": {
        "alarmName": "my-alarm-name",
        "state": {
            "value": "ALARM",
            "reason": "Threshold Crossed: 3 out of the last 3 datapoints [20000.0
 (08/11/22 11:25:00), 20000.0 (08/11/22 11:24:00), 20000.0 (08/11/22 11:23:00)] were
 greater than the threshold (0.0) (minimum 1 datapoint for OK -> ALARM transition).",
            "reasonData": "{\"version\":\"1.0\",\"queryDate\":
\"2022-11-08T11:26:05.399+0000\",\"startDate\":\"2022-11-08T11:23:00.000+0000\",
\"period\":60,\"recentDatapoints\":[20000.0,20000.0,20000.0],\"threshold\":0.0,
\"evaluatedDatapoints\":[{\"timestamp\":\"2022-11-08T11:25:00.000+0000\",\"value
\":20000.0}]}",
            "timestamp": "2022-11-08T11:26:05.401+0000",
            "evaluationState": "PARTIAL_DATA"
        },
        "previousState": {
            "value": "INSUFFICIENT_DATA",
            "reason": "Unchecked: Initial alarm creation",
            "timestamp": "2022-11-08T11:25:51.227+0000"
        },
```

```
        "configuration": {
            "metrics": [
                {
                    "id": "m2",
                    "expression": "SELECT SUM(PartialDataTestMetric) FROM
 partial_data_test",
                    "returnData": true,
                    "period": 60
                }
            ]
        }
    }
}
```

アラームのクエリに、最初に 500 を超える時系列を返す GROUP BY ステートメントが含まれている場合、アラームはクエリで検出された最初の 500 の時系列に基づいて評価されます。ただし、ORDER BY 句を使用すると、クエリで検出されたすべての時系列がソートされます。その中から ORDER BY 句に応じて値が大きい方から、または小さい方から 500 個がアラームの評価に使用されます。

メトリクス数式のクエリで Metrics Insights を使用する

Metrics Insights クエリを、Metric Math 関数への入力として使用することができます。Metric Math 関数の詳細については、「Metric Math を使用する」を参照してください。

GROUP BY 句を含まない Metrics Insights クエリは単一の時系列を返します。この場合に返される結果は、単一の時系列を入力として受け取る任意の Metric Math 関数で使用できます。

Metrics Insights クエリが GROUP BY 句を含む場合には、複数の時系列が返されます。このような結果は、時系列の配列を入力として受け取る任意の Metric Math 関数での使用が可能です。

例えば次のクエリでは、リージョン内にある各バケットにダウンロードされた合計バイト数が、時系列の配列として返されます。

```
SELECT SUM(BytesDownloaded)
FROM SCHEMA("AWS/S3", BucketName, FilterId)
WHERE FilterId = 'EntireBucket'
GROUP BY BucketName
```

コンソール内または GetMetricData オペレーション内のグラフでは、このクエリの結果は q1 として表示されます。このクエリは結果をバイト数で返すので、代わりに MB 単位で結果を表示したい場合は、次の Metric Math 関数を使用します。

```
q1/1024/1024
```

自然言語を使用した CloudWatch Metrics Insights クエリの生成と更新

この機能は、米国東部 (バージニア北部)、米国西部 (オレゴン)、アジアパシフィック (東京) リージョンで CloudWatch 用にプレビューリリース中であり、今後変更される可能性があります。

CloudWatch は自然言語クエリ機能をサポートしており、CloudWatch Metrics Insights と CloudWatch Logs Insights のクエリを生成し、更新する場合に便利です。

この機能を使用すると、どのような CloudWatch データを探しているのかを平易な英語で質問し、具体的に説明できます。自然言語機能が働いて、入力したプロンプトに基づいてクエリが生成され、クエリの仕組みを説明した文が 1 行ずつ表示されます。また、さらにデータを詳しく調査できるように、生成されたクエリを更新することもできます。

お使いの環境によっては、「ネットワーク出力量が最も多い Amazon Elastic Compute Cloud インスタンスはどれですか」や 「実行された読み取り数が上位の 10 個の Amazon DynamoDB テーブルを表示してください」といったプロンプトを入力できます。

この機能を使用して CloudWatch Metrics Insights クエリを生成するには、ビルダーやエディタのビューで CloudWatch Metrics Insights クエリエディタを開き、[クエリを生成] を選択します。

> **Important**
>
> 自然言語クエリ機能を使用するには、CloudWatchFullAccess、CloudWatchReadOnlyAccess、CloudWatchFullAccessV2、Administrato のいずれかのポリシーを使用する必要があります。
> また、新規または既存のカスタマー管理ポリシーまたはインラインポリシーに `cloudwatch:GenerateQuery` アクションを含めることもできます。

クエリの例

このセクションの例では、自然言語機能を使用して、クエリを生成し更新する方法について説明します。

> **Note**
>
> CloudWatch Metrics Insights クエリエディタと構文の詳細については、「CloudWatch Metrics Insights query components and syntax」を参照してください。

例: 自然言語クエリを生成する

自然言語を使用してクエリを生成するには、プロンプトを入力し、[新しいクエリを生成] を選択します。この例は、基本的な検索を実行するクエリを示しています。

プロンプト

次に、読み込みキャパシティーの消費量が多い上位 10 個の DynamoDB テーブルを検索するように指示するプロンプトの例を示します。

```
Show top 10 DynamoDB Tables by consumed reads
```

Query

次に、このプロンプトに基づいて自然言語機能が生成するクエリの例を示します。プロンプトがクエリの前にあるコメントにどのように表示されるかに注意してください。クエリの後に、クエリの仕組みを説明した文があります。

```
# Show top 10 DynamoDB Tables by consumed reads
SELECT SUM("ConsumedReadCapacityUnits")
FROM "AWS/DynamoDB"
GROUP BY TableName
ORDER BY SUM() DESC
LIMIT 10
# This query selects the sum of consumed read capacity units for each DynamoDB table,
 groups the results by table name, orders the results from highest to lowest read
 capacity consumption, and limits the results to the top 10 tables.
```

> **Note**
>
> プロンプトとクエリの仕組みを説明した文が表示されないようにするには、エディタの歯車アイコンを使用します。

例: 自然言語クエリを更新する

生成済みのプロンプトを編集し、[クエリを更新] を選択することで、クエリを更新できます。

プロンプトの更新

次の例は、先ほどのプロンプトを更新したものを示しています。読み込みキャパシティーの消費量が多い上位 10 個の DynamoDB テーブルを検索するのではなく、返されるバイト数で結果をソートするように指示しています。

```
Sort by bytes returned instead
```

クエリの更新

次に、更新したクエリの例を示します。更新したプロンプトが更新したクエリの前にあるコメントにどのように表示されるかに注意してください。クエリの後に、元のクエリがどのように更新されたかを説明した文があります。

```
# Sort by bytes returned instead
SELECT SUM("ReturnedBytes")
FROM "AWS/DynamoDB"
GROUP BY TableName
ORDER BY SUM() DESC
LIMIT 10
# This query modifies the original query to select the sum of returned bytes instead
 of consumed read capacity units, and orders the results from highest to lowest sum of
 returned bytes, limiting the results to the top 10 tables.
```

サービス改善のためのデータ使用をオプトアウトする

AI モデルをトレーニングし、該当するクエリを生成するために提供した自然言語プロンプトデータは、サービスを提供して維持するためにのみ使用されます。例えば、CloudWatch Metrics Insights の品質を高めるために使用される可能性があります。お客様の信頼、プライバシー、コンテンツのセ

キュリティが当社の最優先事項です。詳細については、「AWSサービス規約」と「AWSresponsible AI policy」を参照してください。

自然言語クエリの開発や品質改善に自分のコンテンツが使用されないようにオプトアウトするには、AI サービスオプトアウトポリシーを作成します。クエリ生成機能をはじめすべての CloudWatch AI 機能データ収集をオプトアウトするには、CloudWatch のオプトアウトポリシーを作成する必要があります。詳細については、「AWS Organizations ユーザーガイド」の「AI サービスオプトアウトポリシー」を参照してください。

SQL 推論

CloudWatch Metrics Insights は、特定の SQL クエリの意図を推測するために、いくつかのメカニズムを使用しています。

トピック

- 時間バケット
- フィールド射影
- ORDER BY でのグローバルな集計

時間バケット

クエリから生成された時系列データポイントは、要求された期間に基づいて時間バケットにロールアップされます。標準 SQL で値を集計するには、明示的な GROUP BY 句を定義して、特定の期間におけるすべての観測値をまとめて収集する必要があります。通常は、時系列データをクエリする際はこのような方法が取られます。ただし、CloudWatch Metrics Insights では、時間バケットを推測することで GROUP BY 句を明示的に記述する必要をなくしています。

例えば、1 分の期間を指定しながらクエリを実行すると、その期間中の次の (excluded) までに属するすべての観測値が、タイムバケットの開始時間にロールアップされます。これにより、Metrics Insights での SQL ステートメントがより簡潔になり、冗長性が低くなります。

```
SELECT AVG(CPUUtilization)
FROM SCHEMA("AWS/EC2", InstanceId)
```

前出のクエリでは、すべての Amazon EC2 インスタンスにおける平均 CPU 使用率を表す、単一の時系列 (タイムスタンプと値のペア) を返しています。要求された期間が 1 分であると仮定すると、ここで返される各データポイントは、特定の (開始時間を含み終了時間を除く) 1 分間隔の中で測定

された、すべての観測値の平均を表すことになります。特定のデータポイントに関連するタイムスタンプは、バケットの開始時刻を示します。

フィールド射影

Metrics Insights のクエリは、常にタイムスタンプの射影を返します。SELECT句では、タイムスタンプ列を指定して、対応する各データポイント値のタイムスタンプを取得する必要はありません。タイムスタンプの計算方法の詳細については、「時間バケット」を参照してください。

GROUP BY を使用すると、各グループ名も推論され結果に反映されるので、返された時系列をグループ化することができます。

```
SELECT AVG(CPUUtilization)
FROM SCHEMA("AWS/EC2", InstanceId)
GROUP BY InstanceId
```

前のクエリでは、各 Amazon EC2 インスタンスに関し時系列を返しています。各時系列は、インスタンス ID の値の後にラベル付けされています。

ORDER BY でのグローバルな集計

ORDER BY を使用する場合、FUNCTION() では、順序付けすべき集計関数 (クエリされたメトリクスでのデータポイント値) を推定します。この集計オペレーションは、クエリされた時間ウィンドウ全体において、個々の時系列内で一致したすべてのデータポイントに対して実行されます。

```
SELECT AVG(CPUUtilization)
FROM SCHEMA("AWS/EC2", InstanceId)
GROUP BY InstanceId
ORDER BY MAX()
LIMIT 10
```

前出のクエリでは、結果セットを 10 エントリに制限しながら、各 Amazon EC2 インスタンスの CPU 使用量を返しています。この結果は、要求された時間ウィンドウ内の個々の時系列について、その最大値に基づいて順序付けられます。ORDER BY 句は LIMIT の前に適用されます。したがって、順序付けは 10 個を超える時系列に対して計算されることになります。

Metrics Insights のサンプルクエリ

このセクションでは、コピーして直接使用することも、クエリエディタで編集して使用することもできる、有用な CloudWatch Metrics Insights クエリの例を紹介します。これらの例の内のいくつか

は、コンソールで既に利用可能な状態であり、[Metrics] (メトリクス) ビューで [Add query] (クエリを追加) をクリックしてアクセスすることができます。

Application Load Balancer での例

すべてのロードバランサーでのリクエストの合計数

```
SELECT SUM(RequestCount)
FROM SCHEMA("AWS/ApplicationELB", LoadBalancer)
```

最もアクティブな上位 10 個のロードバランサー

```
SELECT MAX(ActiveConnectionCount)
FROM SCHEMA("AWS/ApplicationELB", LoadBalancer)
GROUP BY LoadBalancer
ORDER BY SUM() DESC
LIMIT 10
```

AWS API の使用状況に関する例

アカウント内の呼び出し数で上位 20 の AWS API

```
SELECT COUNT(CallCount)
FROM SCHEMA("AWS/Usage", Class, Resource, Service, Type)
WHERE Type = 'API'
GROUP BY Service, Resource
ORDER BY COUNT() DESC
LIMIT 20
```

呼び出しごとにソートされた CloudWatch API

```
SELECT COUNT(CallCount)
FROM SCHEMA("AWS/Usage", Class, Resource, Service, Type)
WHERE Type = 'API' AND Service = 'CloudWatch'
GROUP BY Resource
ORDER BY COUNT() DESC
```

DynamoDB での例

実行された読み取り数で上位 10 のテーブル

```
SELECT SUM(ProvisionedWriteCapacityUnits)
FROM SCHEMA("AWS/DynamoDB", TableName)
GROUP BY TableName
ORDER BY MAX() DESC LIMIT 10
```

返されたバイト数で上位 10 のテーブル

```
SELECT SUM(ReturnedBytes)
FROM SCHEMA("AWS/DynamoDB", TableName)
GROUP BY TableName
ORDER BY MAX() DESC LIMIT 10
```

ユーザーエラー数で上位 10 のテーブル

```
SELECT SUM(UserErrors)
FROM SCHEMA("AWS/DynamoDB", TableName)
GROUP BY TableName
ORDER BY MAX() DESC LIMIT 10
```

Amazon Elastic Block Store での例

書き込まれたバイト数で上位 10 のAmazon EBSボリューム

```
SELECT SUM(VolumeWriteBytes)
FROM SCHEMA("AWS/EBS", VolumeId)
GROUP BY VolumeId
ORDER BY SUM() DESC
LIMIT 10
```

Amazon EBS ボリュームでの平均書き込み時間

```
SELECT AVG(VolumeTotalWriteTime)
FROM SCHEMA("AWS/EBS", VolumeId)
```

Amazon EC2 での例

最も高い順にソートされた EC2 インスタンスの CPU 使用量

```
SELECT AVG(CPUUtilization)
  FROM SCHEMA("AWS/EC2", InstanceId)
```

```
  GROUP BY InstanceId
  ORDER BY AVG() DESC
```

フリート全体での平均 CPU 使用量

```
SELECT AVG(CPUUtilization)
FROM SCHEMA("AWS/EC2", InstanceId)
```

CPU の最高使用量で上位 10 のインスタンス

```
SELECT MAX(CPUUtilization)
FROM SCHEMA("AWS/EC2", InstanceId)
GROUP BY InstanceId
ORDER BY MAX() DESC
LIMIT 10
```

このケースでは、CloudWatch エージェントはアプリケーションごとに **CPUUtilization** メトリクスを収集しています。このクエリは、特定のアプリケーション名について、このメトリクスの平均によるフィルタリングを行います。

```
SELECT AVG(CPUUtilization)
FROM "AWS/CWAgent"
WHERE ApplicationName = 'eCommerce'
```

Amazon Elastic Container Service での例

すべての ECS クラスタでの平均 CPU 使用量

```
SELECT AVG(CPUUtilization)
FROM SCHEMA("AWS/ECS", ClusterName)
```

メモリ使用量で上位 10 のクラスタ

```
SELECT AVG(MemoryUtilization)
FROM SCHEMA("AWS/ECS", ClusterName)
GROUP BY ClusterName
ORDER BY AVG() DESC
LIMIT 10
```

CPU 使用量で上位 10 のサービス

```
SELECT AVG(CPUUtilization)
FROM SCHEMA("AWS/ECS", ClusterName, ServiceName)
GROUP BY ClusterName, ServiceName
ORDER BY AVG() DESC
LIMIT 10
```

実行中のタスク (Container Insights) 数で上位 10 のサービス

```
SELECT AVG(RunningTaskCount)
FROM SCHEMA("ECS/ContainerInsights", ClusterName, ServiceName)
GROUP BY ClusterName, ServiceName
ORDER BY AVG() DESC
LIMIT 10
```

Amazon Elastic Kubernetes Service Container Insights での例

すべての EKS クラスタでの平均 CPU 使用量

```
SELECT AVG(pod_cpu_utilization)
FROM SCHEMA("ContainerInsights", ClusterName)
```

ノードの CPU 使用量で上位 10 のクラスタ

```
SELECT AVG(node_cpu_utilization)
FROM SCHEMA("ContainerInsights", ClusterName)
GROUP BY ClusterName
ORDER BY AVG() DESC LIMIT 10
```

ポッドのメモリ使用量で上位 10 のクラスター

```
SELECT AVG(pop_memory_utilization)
FROM SCHEMA("ContainerInsights", ClusterName)
GROUP BY ClusterName
ORDER BY AVG() DESC LIMIT 10
```

CPU 使用量で上位 10 のノード

```
SELECT AVG(node_cpu_utilization)
FROM SCHEMA("ContainerInsights", ClusterName, NodeName)
GROUP BY ClusterName, NodeName
```

```
ORDER BY AVG() DESC LIMIT 10
```

メモリ使用量で上位 10 のポッド

```
SELECT AVG(pod_memory_utilization)
FROM SCHEMA("ContainerInsights", ClusterName, PodName)
GROUP BY ClusterName, PodName
ORDER BY AVG() DESC LIMIT 10
```

EventBridge での例

呼び出し数で上位10 のルール

```
SELECT SUM(Invocations)
FROM SCHEMA("AWS/Events", RuleName)
GROUP BY RuleName
ORDER BY MAX() DESC LIMIT 10
```

失敗した呼び出し数で上位 10 のルール

```
SELECT SUM(FailedInvocations)
FROM SCHEMA("AWS/Events", RuleName)
GROUP BY RuleName
ORDER BY MAX() DESC LIMIT 10
```

一致したルール数で上位 10 のルール

```
SELECT SUM(MatchedEvents)
FROM SCHEMA("AWS/Events", RuleName)
GROUP BY RuleName
ORDER BY MAX() DESC LIMIT 10
```

Kinesis での例

書き込まれたバイト数で上位 10 のストリーム

```
SELECT SUM("PutRecords.Bytes")
FROM SCHEMA("AWS/Kinesis", StreamName)
GROUP BY StreamName
ORDER BY SUM() DESC LIMIT 10
```

ストリーム内の最も早いアイテム数で上位 10 のストリーム

```
SELECT MAX("GetRecords.IteratorAgeMilliseconds")
FROM SCHEMA("AWS/Kinesis", StreamName)
GROUP BY StreamName
ORDER BY MAX() DESC LIMIT 10
```

Lambda での例

呼び出し数で順序付けられた Lambda 関数

```
SELECT SUM(Invocations)
FROM SCHEMA("AWS/Lambda", FunctionName)
GROUP BY FunctionName
ORDER BY SUM() DESC
```

最長実行時間で上位 10 の Lambda 関数

```
SELECT AVG(Duration)
FROM SCHEMA("AWS/Lambda", FunctionName)
GROUP BY FunctionName
ORDER BY MAX() DESC
LIMIT 10
```

エラー発生数で上位 10 のLambda 関数

```
SELECT SUM(Errors)
FROM SCHEMA("AWS/Lambda", FunctionName)
GROUP BY FunctionName
ORDER BY SUM() DESC
LIMIT 10
```

CloudWatch Logs での例

受信イベント数で上位 10 のロググループ

```
SELECT SUM(IncomingLogEvents)
FROM SCHEMA("AWS/Logs", LogGroupName)
GROUP BY LogGroupName
ORDER BY SUM() DESC LIMIT 10
```

書き込まれたバイト数で上位 10 のロググループ

```
SELECT SUM(IncomingBytes)
FROM SCHEMA("AWS/Logs", LogGroupName)
GROUP BY LogGroupName
ORDER BY SUM() DESC LIMIT 10
```

Amazon RDS での例

CPU の最大使用量で上位 10 の Amazon RDS インスタンス

```
SELECT MAX(CPUUtilization)
FROM SCHEMA("AWS/RDS", DBInstanceIdentifier)
GROUP BY DBInstanceIdentifier
ORDER BY MAX() DESC
LIMIT 10
```

書き込み量で上位 10 の Amazon RDS クラスター

```
SELECT SUM(WriteIOPS)
FROM SCHEMA("AWS/RDS", DBClusterIdentifier)
GROUP BY DBClusterIdentifier
ORDER BY MAX() DESC
LIMIT 10
```

Amazon Simple Storage Service での例

バケット別の平均レイテンシー

```
SELECT AVG(TotalRequestLatency)
FROM SCHEMA("AWS/S3", BucketName, FilterId)
WHERE FilterId = 'EntireBucket'
GROUP BY BucketName
ORDER BY AVG() DESC
```

ダウンロードされたバイト数で上位 10 のバケット

```
SELECT SUM(BytesDownloaded)
FROM SCHEMA("AWS/S3", BucketName, FilterId)
WHERE FilterId = 'EntireBucket'
GROUP BY BucketName
ORDER BY SUM() DESC
```

```
LIMIT 10
```

Amazon Simple Notification Service での例

SNS トピックによって発行されたメッセージの総数

```
SELECT SUM(NumberOfMessagesPublished)
FROM SCHEMA("AWS/SNS", TopicName)
```

発行されたメッセージ数で上位 10 のトピック

```
SELECT SUM(NumberOfMessagesPublished)
FROM SCHEMA("AWS/SNS", TopicName)
GROUP BY TopicName
ORDER BY SUM() DESC
LIMIT 10
```

メッセージ配信の失敗数で上位 10 のトピック

```
SELECT SUM(NumberOfNotificationsFailed)
FROM SCHEMA("AWS/SNS", TopicName)
GROUP BY TopicName
ORDER BY SUM() DESC
LIMIT 10
```

Amazon SQS での例

表示可能なメッセージ数で上位 10 のキュー

```
SELECT AVG(ApproximateNumberOfMessagesVisible)
FROM SCHEMA("AWS/SQS", QueueName)
GROUP BY QueueName
ORDER BY AVG() DESC
LIMIT 10
```

最もアクティブな上位 10 のキュー

```
SELECT SUM(NumberOfMessagesSent)
FROM SCHEMA("AWS/SQS", QueueName)
GROUP BY QueueName
ORDER BY SUM() DESC
```

```
LIMIT 10
```

最も初期のメッセージからの経過時間で上位 10 のキュー

```
SELECT AVG(ApproximateAgeOfOldestMessage)
FROM SCHEMA("AWS/SQS", QueueName)
GROUP BY QueueName
ORDER BY AVG() DESC
LIMIT 10
```

Metrics Insights の制限

CloudWatch Metrics Insights には現在、次の制限があります。

- 現在、クエリが可能なのは直近 3 時間のデータのみです。
- 1 つのクエリで処理できるメトリクスは 10,000 個以下です。これは、SELECT、FROM、および WHERE 句で 10,000 個を超えるメトリクスが一致した場合でも、クエリは、これらの検出したメトリクスのうち最初の 10,000 個のみを処理することを意味します。
- 1 つのクエリで 500 を超える時系列を返すことはできません。これは、クエリが 500 個を超えるメトリクスを返すよう指定した場合でも、クエリ結果にすべてのメトリクスが返されるわけではないことを意味します。ORDER BY 句を使用すると、処理されているすべてのメトリクスがソートされます。その中から (ORDER BY 句に応じ) 最大値または最小値を持つ 500 個が返されます。

  ORDER BY 句を含めない場合は、一致したメトリクスの中から、どの 500 個が選択され返されるのかを制御することはできません。
- 各リージョンで最大 200 個の Metrics Insights アラームを設定できます。
- Metrics Insights では、高解像度データ (1 分未満の精度で報告されるメトリクスデータ) はサポートされていません。高解像度データをリクエストしてもリクエストは失敗しませんが、出力は 1 分単位で集計されます。
- 個別の GetMetricData オペレーションに含めることができるクエリは 1 つだけですが、ダッシュボードには、それぞれにクエリを 1 つ含んだ複数のウィジェットを用意することができます。

Metrics Insights 用語集

ラベル

Metrics Insights では、ラベルはキーと値のペアであり、特定のデータセットを返すようにクエリの範囲を指定するために使用します。また、クエリ結果を個別の時系列に分割する際の基準

を定義するためにも使用できます。ラベルキーは SQL での列名に似ています。現在、ラベルは CloudWatch メトリクスのディメンションである必要があります。

監視

観測値とは、特定の時間に特定のメトリクスについて記録された値です。

Metrics Insights のトラブルシューティング

結果に、実際には使用していない「Other」というディメンションが含まれている

これは、このクエリに含まれる GROUP BY 句で指定されているラベルキーが、このクエリによって返されるメトリクスの一部で使用されていないことを意味します。この場合、Other という名前の NULL グループは返されません。そのラベルキーを含まないメトリクスは、おそらく、そのラベルキーの値すべてから集約された値を返す、集計メトリクスです。

例えば、次のようなクエリがあるとします。

```
SELECT AVG(Faults)
FROM MyCustomNamespace
GROUP BY Operation, ServiceName
```

返されたメトリクスの一部に、ディメンションとして ServiceName が含まれない場合、これらのメトリクスは、ServiceName の値として Other が割り当てられて表示されます。

結果に「Other」が表示されないようにするには、FROM 句で SCHEMA を使用します。以下にその例を示します。

```
SELECT AVG(Faults)
FROM SCHEMA(MyCustomNamespace, Operation)
GROUP BY Operation, ServiceName
```

これにより、結果として返されるメトリクスは、Operation と ServiceName 両方のディメンションを持つもののみに制限されます。

グラフ内の最も古いタイムスタンプは、他のメトリクスよりも低いメトリクス値を持っています

CloudWatch Metrics Insights は現在、最新 3 時間のデータのみをサポートしています。1 分を超える周期でグラフを作成する場合、最も古いデータポイントが、想定される値とは異なることがありま

す。これは、Metrics Insights のクエリが最新の 3 時間分のデータのみを返すためです。この場合、クエリ内の最も古いデータポイントは、そのデータポイントの期間内のすべての観測値を返すのではなく、過去 3 時間の境界内に測定された観測値のみを返します。

メトリクスエクスプローラーを使用して、タグとプロパティ別にリソースをモニタリングする

メトリクスエクスプローラーは、タグおよびリソースプロパティ別にメトリクスをフィルタリング、集計、および可視化できるタグベースのツールです。これにより、サービスのモニタリング性を高めることができます。これを使用することで、柔軟で動的なトラブルシューティングエクスペリエンスが得られるため、一度に複数のグラフを作成し、これらのグラフを使用してアプリケーション正常性ダッシュボードを構築できます。

メトリクスエクスプローラーの可視化は動的であるため、メトリクスエクスプローラーウィジェットを作成して CloudWatch ダッシュボードに追加した後に一致するリソースを作成すると、新しいリソースがエクスプローラーウィジェットに自動的に表示されます。

例えば、すべての EC2 本稼働用インスタンスに **`production`** タグが付いている場合、メトリクスエクスプローラーを使用して、これらのすべてのインスタンスからメトリクスをフィルタリングして集計し、その正常性とパフォーマンスを把握できます。一致するタグを持つ新しいインスタンスが後で作成されると、メトリクスエクスプローラーウィジェットに自動的に追加されます。

> **Note**
>
> Metrics explorer は特定の時点のエクスペリエンスを提供します。終了したリソース、または指定したプロパティやタグで存在しなくなったリソースは、ビジュアライゼーションには表示されません。ただし、これらのリソースのメトリクスは CloudWatch メトリクスビューで引き続き確認できます。

メトリクスエクスプローラーを使用すると、基準に一致するリソースからメトリクスを集計する方法と、1 つのメトリクスエクスプローラーウィジェット内ですべてのメトリクスを 1 つのグラフで表示するか、異なるグラフで表示するかを選択できます。

メトリクスエクスプローラーには、便利なビジュアライゼーショングラフをワンクリックで表示するために使用できるテンプレートが含まれています。また、これらのテンプレートを拡張して、完全にカスタマイズされたメトリクスエクスプローラーウィジェットを作成することもできます。

メトリクスエクスプローラーでは、AWS によって出力されるメトリクスと CloudWatch エージェントによって発行される EC2 メトリクス (メモリ、ディスク、CPU メトリクスなど) がサポートされます。メトリクスエクスプローラーを使用して、CloudWatch エージェントによって発行されているメトリクスを確認するには、CloudWatch エージェント設定ファイルを更新する必要がある場合があります。詳細については、「メトリクスエクスプローラーの CloudWatch エージェント設定」を参照してください。

メトリクスエクスプローラーを使用してビジュアライゼーションを作成し、必要に応じてダッシュボードに追加するには、次の手順を実行します。

メトリクスエクスプローラーでビジュアライゼーションを作成するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで、[Explorer] (エクスプローラー) を選択します。
3. 以下のいずれかを実行します。
   - テンプレートを使用するには、現在 [Empty Explorer] (空のエクスプローラー) が表示されているボックスでテンプレートを選択します。

     テンプレートによっては、エクスプローラーにメトリクスのグラフがすぐに表示される場合があります。表示されない場合は、[From] (開始) ボックスで 1 つまたは複数のタグまたはプロパティを選択すると、データが表示されます。表示されない場合は、ページの上部にあるオプションを使用して、グラフでより長い時間範囲を表示するように設定します。
   - カスタムビジュアライゼーションを作成するには、[Metrics] (メトリクス) で、サービスから 1 つのメトリクスまたは使用可能なすべてのメトリクスを選択します。

     メトリクスを選択した後、必要に応じてこのステップを繰り返して、さらにメトリクスを追加できます。
4. 選択した各メトリクスについて、CloudWatch は、メトリクス名の直後に使用する統計を表示します。これを変更するには、統計名を選択し、必要な統計を選択します。
5. [From] (開始) で、結果をフィルタリングするためのタグまたはリソースプロパティを選択します。

   これを実行した後、必要に応じてこのステップを繰り返して、さらにタグまたはリソースプロパティを選択できます。

   2 つの EC2 インスタンスタイプなど、同じプロパティの複数の値を選択した場合、Explorer には、選択したいずれかのプロパティに一致するすべてのリソースが表示されます。OR 演算として取り扱われます。

Production タグや M5 インスタンスタイプなど、異なるプロパティまたはタグを選択した場合、これらのすべての選択に一致するリソースのみが表示されます。これは、AND 演算として扱われます。

6. (オプション) [Aggregate by] (集計単位) で、メトリクスの集計に使用する統計を選択します。その後、[for] (次のため:) の横で、リストからメトリクスの集計方法を選択します。現在表示されているすべてのリソースを集計することも、単一のタグまたはリソースプロパティ別に集計することもできます。

   集計方法に応じて、結果は 1 つの時系列または複数の時系列になります。

7. [Split by] (分割方法) では、複数の時系列を持つ単一のグラフを複数のグラフに分割できます。分割は、[Split by] (分割方法) で選択するさまざまな基準によって行うことができます。
8. [Graph options] (グラフのオプション) で、期間、グラフのタイプ、凡例の配置、およびレイアウトを変更することにより、より洗練されたグラフを表示できます。
9. このビジュアライゼーションをウィジェットとして CloudWatch ダッシュボードに追加するには、[Add to dashboard] (ダッシュボードに追加) を選択します。

メトリクスエクスプローラーの CloudWatch エージェント設定

メトリクスエクスプローラーで CloudWatch エージェントによって発行された EC2 メトリクスを検出できるようにするには、CloudWatch エージェント設定ファイルに次の値が含まれていることを確認します。

- `metrics` セクションで、`aggregation_dimensions` パラメータに `["InstanceId"]` が含まれていることを確認します。また、他のディメンションを含めることもできます。
- `metrics` セクションで、`append_dimensions` パラメータに `{"InstanceId":"${aws:InstanceId}"}` 行が含まれていることを確認します。また、他の行を含めることもできます。
- `metrics` セクションの `metrics_collected` セクション内で、メトリクスエクスプローラーで検出する各リソースタイプ (`cpu`、`disk`、および `memory` セクションなど) のセクションを確認します。これらの各セクションに `"resources": [ "*"] line.` が含まれていることを確認します。
- `metrics_collected` セクションの `cpu` セクションで、`"totalcpu": true` 行があることを確認します。
- CloudWatch エージェントによって収集されるメトリクスには、カスタム名前空間ではなく、デフォルトの `CWAgent` 名前空間を使用する必要があります。

前のリストの設定により、CloudWatch エージェントは、ディスク、CPU、およびその他のリソース (メトリクスエクスプローラーでプロットでき、それを使用するすべてのインスタンスについてのもの) の集計メトリクスを発行します。

これらの設定は、以前に複数のディメンションで発行するように設定したメトリクスを再発行し、メトリクスのコストに加算します。

CloudWatch エージェント設定ファイルの編集の詳細については、「CloudWatch エージェント設定ファイルを手動で作成または編集する」を参照してください。

メトリクスストリームを使用する

メトリクスストリームを使用すると、ほぼリアルタイムに着信するように低レイテンシーで、選択した送信先に CloudWatch メトリクスを継続的にストリーミングできます。サポートされる送信先には、Amazon Simple Storage Service などの AWS の送信先や、サードパーティーのサービスプロバイダーの送信先が含まれます。

CloudWatch メトリクスストリームには、主に次の 3 つの使用シナリオがあります。

- Firehose を使用したカスタムセットアップ – メトリクスストリームを作成し、それを (CloudWatch メトリクスを必要な場所に配信するための) Amazon Data Firehose 配信ストリームに転送します。これらは、Amazon S3 などのデータレイク、または、サードパーティプロバイダーを含め Firehose でサポートされる任意の送信先またはエンドポイントにストリーミングすることができます。ネイティブでサポートされるのは JSON 形式、OpenTelemetry 1.0.0 形式、OpenTelemetry 0.7.0 形式です。また、Firehose 配信ストリームで変換を設定すると、データを Parquet などの別の形式に変換することもできます。メトリクスストリームにより、モニタリングデータの継続的な更新ができます。あるいは、この CloudWatch メトリクスデータを請求やパフォーマンスに関するデータと組み合わせることで、豊富なデータセットを作成できます。その後、Amazon Athena などのツールを使用して、コストの最適化、リソースのパフォーマンス、およびリソースの使用率に関するインサイトを得ることができます。
- クイック S3 セットアップ – 簡単なプロセスで Amazon Simple Storage Service へのストリーミングを設定します。デフォルトでは、ストリームに必要なリソースを CloudWatch が作成します。サポートされている形式は、JSON、OpenTelemetry 1.0.0、OpenTelemetry 0.7.0 です。
- AWS パートナーのクイックセットアップ – CloudWatch では、一部のサードパーティパートナー向けとして、素早い設定エクスペリエンスを用意しています。CloudWatch ストリーミングデータを使用したアプリケーションのモニタリング、トラブルシューティング、および分析のために、サードパーティーのサービスプロバイダーを利用できます。クイックパートナーセットアップの

ワークフローを使用すれば、ユーザーは送信先の URL と API キーを指定するだけで、残りの設定は CloudWatch が処理します。クイックパートナーセットアップは、次のサードパーティープロバイダーで利用できます。

- Datadog
- Dynatrace
- New Relic
- Splunk Observability Cloud
- SumoLogic

すべての CloudWatch メトリクスをストリーミングすることも、フィルターを使用して特定のメトリクスだけをストリーミングすることもできます。各メトリクスストリームには、メトリクス名前空間または特定のメトリクスを含めるか除外するためのフィルターを、最大 1000 個含めることができます。1 つのメトリクスストリームには、含めるフィルターまたは除外するフィルターのみを設定できますが、両方を含めることはできません。

メトリクスストリームが作成された後、適用されているフィルターに適合する新しいメトリクスが作成されると、新しいメトリクスは自動的にストリームに含まれます。

アカウントごとまたはリージョンごとのメトリクスストリーム数に制限はなく、ストリーミングされるメトリクス更新の数にも制限はありません。

各ストリームは、JSON、OpenTelemetry 1.0.0、OpenTelemetry 0.7.0 のいずれの形式でも構いません。メトリクスストリームの出力形式は、OpenTelemetry 0.7.0 から OpenTelemetry 1.0.0 にアップグレードする場合などいつでも編集できます。出力形式の詳細については、「メトリクスストリームの出力形式」を参照してください。

モニタリングアカウントのメトリクストリームについては、そのモニタリングアカウントにリンクされているソースアカウントからのメトリクスを含めるかどうかを選択できます。詳細については、「CloudWatch のクロスアカウントオブザーバビリティ」を参照してください。

メトリクスストリームには、常に `Minimum`、`Maximum`、`SampleCount`、および `Sum` の統計情報が含まれています。また、追加料金で、追加の統計情報を含めることもできます。詳細については、「ストリーミングできる統計情報」を参照してください。

メトリクスストリームの料金は、メトリクス更新の件数によります。メトリクスストリームに使用される配信ストリームについても、Firehose からの料金が発生します。詳細については、「Amazon CloudWatch 料金表」をご覧ください。

トピック

メトリクスストリームをセットアップする

以下のセクションの手順を使用して、CloudWatch メトリクスストリームを設定します。

メトリクスストリームが作成された後、メトリクスデータが送信先に表示されるまでにかかる時間は、Firehose 配信ストリームに設定されているバッファリング設定によって異なります。バッファリングは、最大ペイロードサイズまたは最大待機時間のいずれか最初に到達した方という形で表されます。これらの値が最小値 (60 秒、1 MB) に設定されている場合、選択した CloudWatch 名前空間にアクティブなメトリクス更新がある場合、予想されるレイテンシーは 3 分以内です。

CloudWatch メトリクスストリームでは、データは 1 分ごとに送信されます。データは最終送信先に順番通りに到着しない可能性があります。タイムスタンプからの経過時間が 3 日以上前のタイムスタンプを持つメトリクスを除き、指定した名前空間のすべての指定メトリクスがメトリクスストリームに送信されます。

ストリーミングするメトリクス名と名前空間の組み合わせごとに、そのメトリクス名と名前空間のすべてのディメンションの組み合わせがストリーミングされます。

モニタリングアカウントのメトリクストリームについては、そのモニタリングアカウントにリンクされているソースアカウントからのメトリクスを含めるかどうかを選択できます。詳細については、「CloudWatch のクロスアカウントオブザーバビリティ」を参照してください。

メトリクスストリームを作成および管理するには、 CloudWatchFullAccess ポリシーと `iam:PassRole` アクセス許可を持つアカウント、または次のリストのアクセス許可を持つアカウントにログオンする必要があります。

- `iam:PassRole`
- `cloudwatch:PutMetricStream`

- `cloudwatch:DeleteMetricStream`
- `cloudwatch:GetMetricStream`
- `cloudwatch:ListMetricStreams`
- `cloudwatch:StartMetricStreams`
- `cloudwatch:StopMetricStreams`

CloudWatch でメトリクスストリームに必要な IAM ロールを設定する場合は、`iam:CreateRole` および `iam:PutRolePolicy` アクセス許可も必要です。

> **Important**
>
> `cloudwatch:PutMetricStream` を持つユーザは、 `cloudwatch:GetMetricData` アクセス許可がない場合でも、ストリーミングされる CloudWatch メトリクスデータにアクセスできます。

トピック

- Firehose でのカスタム設定
- Amazon S3 のクイックセットアップを使用する
- クイックパートナーセットアップ

Firehose でのカスタム設定

このメソッドを使用してメトリクスストリームを作成し、それを (CloudWatch メトリクスを必要な場所に配信する) Amazon Data Firehose 配信ストリームに転送します。これらは、Amazon S3 などのデータレイク、または、サードパーティプロバイダーを含め Firehose でサポートされる任意の送信先またはエンドポイントにストリーミングすることができます。

ネイティブでサポートされるのは JSON 形式、OpenTelemetry 1.0.0 形式、OpenTelemetry 0.7.0 形式です。また、Firehose 配信ストリームで変換を設定すると、データを Parquet などの別の形式に変換することもできます。メトリクスストリームにより、モニタリングデータの継続的な更新ができます。あるいは、この CloudWatch メトリクスデータを請求やパフォーマンスに関するデータと組み合わせることで、豊富なデータセットを作成できます。その後、Amazon Athena などのツールを使用して、コストの最適化、リソースのパフォーマンス、およびリソースの使用率に関するインサイトを得ることができます。

CloudWatch コンソール、AWS CLI、AWS CloudFormation、または AWS Cloud Development Kit (AWS CDK) を使用して、メトリクスストリームを設定できます。

メトリクスストリームに使用する Firehose 配信ストリームは、メトリクスストリームを設定したのと同じアカウントかつ同じリージョンに存在する必要があります。別のアカウントまたは別のリージョンにある最終配信先にストリーミングするように Firehose 配信ストリームを設定すれば、クロスリージョン機能を実現できます。

CloudWatch コンソール

このセクションでは、CloudWatch コンソールで Firehose を使用したメトリクスストリームの設定方法について説明します。

Firehose を使用してカスタムメトリクスストリームを設定するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで、 [Metrics] (メトリクス)、[Streams] (ストリーム) の順に選択します。[Create metric stream] (メトリクスストリームの作成) を選択します。
3. (オプション) CloudWatch クロスアカウントオブザーバビリティのモニタリングアカウントとしてセットアップされたアカウントにサインインしている場合は、リンクされたソースアカウントからのメトリクスをこのメトリクスストリームに含めるかどうかを選択できます。ソースアカウントからのメトリクスを含めるには、[Include source account metrics] (ソースアカウントメトリクスを含める) を選択します。
4. [Firehose でカスタム設定] を選択します。
5. [Kinesis Data Firehose ストリームを選択する] で、使用する Firehose 配信ストリームを選択します。同じアカウントに存在する必要があります。このオプションのデフォルトの形式は OpenTelemetry 0.7.0 ですが、後でこの形式を変更することもできます。

   次に、[Firehose 配信ストリームを選択する] で、使用する Firehose 配信ストリームを選択します。
6. (オプション) CloudWatch に新しいロールを作成させる代わりに、[既存のサービスロールを選択] を選択して既存の IAM ロールを使用できます。
7. (オプション) シナリオのデフォルトの形式から出力形式を変更するには、[出力形式を変更] を選択します。サポートされている形式は、JSON、OpenTelemetry 1.0.0、OpenTelemetry 0.7.0 です。
8. メトリクスをストリーミングするには、[すべてのメトリクス] または [メトリクスを選択] を選択します。

[すべてのメトリクス] を選択した場合には、このアカウントのすべてのメトリクスがストリームに含まれます。

すべてのメトリクスをストリーミングするかどうかを慎重に検討してください。ストリーミングするメトリクスが多いほど、メトリクスストリームの料金は高くなります。

[メトリクスを選択] を選択した場合には、以下のいずれかの操作を行います。

- ほとんどのメトリクス名前空間をストリーミングするには、[除外] を選択し、除外する名前空間またはメトリクスを選択します。[除外] で名前空間を指定すると、オプションで、除外する特定のメトリクスをその名前空間から選択できます。メトリクスを選択せずに名前空間の除外のみを選択した場合には、その名前空間からすべてのメトリクスが除外されます。
- メトリクスストリームにメトリクス名前空間またはメトリクスをいくつか含めるには、[含める] を選択し、含める名前空間またはメトリクスを選択します。メトリクスを選択せずに名前空間を含めるよう選択した場合には、その名前空間のすべてのメトリクスが含まれます。

9. (オプション) これらのメトリクスの一部について、Minimum (最小値)、Maximum (最大値)、SampleCount (サンプル数)、Sum (合計) 以外の追加の統計情報をストリーミングするには、[統計をさらに追加] を選択します。[Add recommended metrics] (推奨メトリクスを追加する) を選択してよく使用される統計情報を追加するか、名前空間とメトリクス名を手動で選択して追加の統計情報をストリーミングするかのどちらかを行います。次に、ストリーミングする追加の統計情報を選択します。

   別の追加の統計情報のセットをストリーミングする別のメトリクスグループを選択するには、[Add additional statistics] (統計情報の追加) を選択します。各メトリクスには最大 20 個の追加の統計情報を含めることができ、メトリクスストリーム内の最大 100 個のメトリクスに追加の統計情報を含めることができます。

   追加の統計情報をストリーミングすると、より多くの料金が発生します。詳細については、「ストリーミングできる統計情報」を参照してください。

   追加の統計情報の定義については、「CloudWatch 統計定義」を参照してください。

10. (オプション) [メトリクスストリーム名] で新しいメトリクスストリームの名前をカスタマイズします。

11. [メトリクスストリームの作成] を選択します。

AWS CLI または AWS API

CloudWatch メトリクスストリームを作成するには、次の手順に従います。

AWS CLI または AWS API を使用してメトリクスストリームを作成するには

1. Amazon S3 にストリーミングする場合は、まずバケットを作成します。詳細については、「バケットの作成」を参照してください。
2. Firehose 配信ストリームを作成します。詳細については、「Creating a Firehose stream」をご参照ください。
3. CloudWatch が Firehose 配信ストリームに書き込むことを可能にする IAM ロールを作成します。このロールの内容の詳細については、「CloudWatch と Firehose 間の信頼」を参照してください。
4. `aws cloudwatch put-metric-stream` CLI コマンドまたは `PutMetricStream` API を使用して、CloudWatch メトリクスストリームを作成します。

AWS CloudFormation

AWS CloudFormation を使用して、メトリクスストリームを設定できます。詳細については、「AWS::CloudWatch::MetricStream」を参照してください。

AWS CloudFormation を使用してメトリクスストリームを作成するには

1. Amazon S3 にストリーミングする場合は、まずバケットを作成します。詳細については、「バケットの作成」を参照してください。
2. Firehose 配信ストリームを作成します。詳細については、「Creating a Firehose stream」をご参照ください。
3. CloudWatch が Firehose 配信ストリームに書き込むことを可能にする IAM ロールを作成します。このロールの内容の詳細については、「CloudWatch と Firehose 間の信頼」を参照してください。
4. AWS CloudFormation でストリームを作成します。詳細については、「AWS::CloudWatch::MetricStream」を参照してください。

AWS Cloud Development Kit (AWS CDK)

AWS Cloud Development Kit (AWS CDK) を使用して、メトリクスストリームを設定できます。

AWS CDK を使用してメトリクスストリームを作成するには

1. Amazon S3 にストリーミングする場合は、まずバケットを作成します。詳細については、「バケットの作成」を参照してください。
2. Firehose 配信ストリームを作成します。詳細については、「Creating an Amazon Data Firehose Delivery Stream」を参照してください。
3. CloudWatch が Firehose 配信ストリームに書き込むことを可能にする IAM ロールを作成します。このロールの内容の詳細については、「CloudWatch と Firehose 間の信頼」を参照してください。
4. メトリクスストリームを作成します。メトリクスストリームのリソースは、`CfnMetricStream` という名前の Level 1 (L1) Constructとして AWS CDK で使用できます。詳細については、「L1 Construct の使用」を参照してください。

Amazon S3 のクイックセットアップを使用する

サポートされている JSON、OpenTelemetry 1.0.0、OpenTelemetry 0.7.0 以外の形式に変換する必要がない場合は、クイック S3 セットアップメソッドを使用すると、Amazon S3 へのストリームをすばやくセットアップできます。CloudWatch は、Firehose 配信ストリームや必要な IAM ロールを含む、必要なすべてのリソースを作成します。このオプションのデフォルトの形式は JSON ですが、ストリームの設定時に形式を変更することができます。

また、最終的な形式を Parquet または Optimized Row Columnar (ORC) にする場合は、「Firehose でのカスタム設定」の手順に従ってください。

CloudWatch コンソール

このセクションでは、CloudWatch コンソールでクイック S3 セットアップを使用して、Amazon S3 のメトリクスストリームを設定する方法について説明します。

クイック S3 セットアップ を使用してメトリクスストリームを設定するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで、 [Metrics] (メトリクス)、[Streams] (ストリーム) の順に選択します。[Create metric stream] (メトリクスストリームの作成) を選択します。
3. (オプション) CloudWatch クロスアカウントオブザーバビリティのモニタリングアカウントとしてセットアップされたアカウントにサインインしている場合は、リンクされたソースアカウントからのメトリクスをこのメトリクスストリームに含めるかどうかを選択できます。ソースアカウ

ントからのメトリクスを含めるには、[Include source account metrics] (ソースアカウントメトリクスを含める) を選択します。

4. [クイック S3 セットアップ] を選択します。CloudWatch は、Firehose 配信ストリームや必要な IAM ロールを含む、必要なすべてのリソースを作成します。このオプションのデフォルトの形式は JSON ですが、後でこの形式を変更できます。
5. (オプション) CloudWatch に新しいリソースを作成させる代わりに、[既存のリソースを選択] を選択すれば、既存の S3 バケットか既存の IAM ロールを使用することができます。
6. (オプション) シナリオのデフォルトの形式から出力形式を変更するには、[出力形式を変更] を選択します。サポートされている形式は、JSON、OpenTelemetry 1.0.0、OpenTelemetry 0.7.0 です。
7. メトリクスをストリーミングするには、[すべてのメトリクス] または [メトリクスを選択] を選択します。

   [すべてのメトリクス] を選択した場合には、このアカウントのすべてのメトリクスがストリームに含まれます。

   すべてのメトリクスをストリーミングするかどうかを慎重に検討してください。ストリーミングするメトリクスが多いほど、メトリクスストリームの料金は高くなります。

   [メトリクスを選択] を選択した場合には、以下のいずれかの操作を行います。

   - ほとんどのメトリクス名前空間をストリーミングするには、[除外] を選択し、除外する名前空間またはメトリクスを選択します。[除外] で名前空間を指定すると、オプションで、除外する特定のメトリクスをその名前空間から選択できます。メトリクスを選択せずに名前空間の除外のみを選択した場合には、その名前空間からすべてのメトリクスが除外されます。
   - メトリクスストリームにメトリクス名前空間またはメトリクスをいくつか含めるには、[含める] を選択し、含める名前空間またはメトリクスを選択します。メトリクスを選択せずに名前空間を含めるよう選択した場合には、その名前空間のすべてのメトリクスが含まれます。
8. (オプション) これらのメトリクスの一部について、Minimum (最小値)、Maximum (最大値)、SampleCount (サンプル数)、Sum (合計) 以外の追加の統計情報をストリーミングするには、[統計をさらに追加] を選択します。[Add recommended metrics] (推奨メトリクスを追加する) を選択してよく使用される統計情報を追加するか、名前空間とメトリクス名を手動で選択して追加の統計情報をストリーミングするかのどちらかを行います。次に、ストリーミングする追加の統計情報を選択します。

別の追加の統計情報のセットをストリーミングする別のメトリクスグループを選択するには、[Add additional statistics] (統計情報の追加) を選択します。各メトリクスには最大 20 個の追加の統計情報を含めることができ、メトリクスストリーム内の最大 100 個のメトリクスに追加の統計情報を含めることができます。

追加の統計情報をストリーミングすると、より多くの料金が発生します。詳細については、「ストリーミングできる統計情報」を参照してください。

追加の統計情報の定義については、「CloudWatch 統計定義」を参照してください。

9. (オプション) [メトリクスストリーム名] で新しいメトリクスストリームの名前をカスタマイズします。
10. [メトリクスストリームの作成] を選択します。

クイックパートナーセットアップ

CloudWatch では、以下のサードパーティパートナー向けとして、クイックセットアップのエクスペリエンスを提供しています。このワークフローを使用するのに指定の必要があるのは、送信先のための URL と API キーのみです。Firehose 配信ストリームや必要な IAM ロールの作成など、その他のセットアップは、CloudWatch により処理されます。

> **Important**
>
> クイックパートナーセットアップを使用してメトリクスストリームを作成する前に、以下のリストからリンクされている各パートナーのドキュメントを読むことを強くお勧めします。

- Datadog
- Dynatrace
- New Relic
- Splunk Observability Cloud
- SumoLogic

これらのパートナーのいずれかに対してメトリクスストリームを設定すると、そのストリームは、以下のセクションで一覧されているようなデフォルト設定を使用して作成されます。

トピック

クイックパートナーセットアップを使用してメトリクスストリームを設定する

CloudWatch では、一部のサードパーティパートナー向けとしてクイックセットアップのオプションを提供しています。このセクションに記載されている手順を開始する前に、対象のパートナーについて一定の情報を確認する必要があります。通常こういった情報には、パートナー側の送信先に関する URL や API キーが含まれています。また、前のセクションからリンクしているパートナーのウェブサイトにあるドキュメントと、次のセクションに一覧されている、そのパートナーでのデフォルト設定も確認します。

Quick Setup でサポートされていないサードパーティを送信先としてストリーミングを行うには、「Firehose でのカスタム設定」の手順に従い、Firehose を使用してストリームをセットアップした後、Firehose から最終的な送信先に対象のメトリクスを送信します。

クイックパートナーセットアップを使用してサードパーティプロバイダーへのメトリクスストリームを作成するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで、 [Metrics] (メトリクス)、[Streams] (ストリーム) の順に選択します。[Create metric stream] (メトリクスストリームの作成) を選択します。
3. (オプション) CloudWatch クロスアカウントオブザーバビリティのモニタリングアカウントとしてセットアップされたアカウントにサインインしている場合は、リンクされたソースアカウントからのメトリクスをこのメトリクスストリームに含めるかどうかを選択できます。ソースアカウントからのメトリクスを含めるには、[Include source account metrics] (ソースアカウントメトリクスを含める) を選択します。
4. [Quick Amazon Web Services partner setup] を選択します。
5. メトリクスのストリーミング先となるパートナーの名前を選択します。
6. [エンドポイント URL] には、送信先の URL を入力します。
7. [アクセスキー] または [API キー] には、パートナーのアクセスキーを入力します。すべてのパートナーにアクセスキーが必要なわけではありません。

8. メトリクスをストリーミングするには、[すべてのメトリクス] または [メトリクスを選択] を選択します。

   [すべてのメトリクス] を選択した場合には、このアカウントのすべてのメトリクスがストリームに含まれます。

   すべてのメトリクスをストリーミングするかどうかを慎重に検討してください。ストリーミングするメトリクスが多いほど、メトリクスストリームの料金は高くなります。

   [メトリクスを選択] を選択した場合には、以下のいずれかの操作を行います。

   - ほとんどのメトリクス名前空間をストリーミングするには、[除外] を選択し、除外する名前空間またはメトリクスを選択します。[除外] で名前空間を指定すると、オプションで、除外する特定のメトリクスをその名前空間から選択できます。メトリクスを選択せずに名前空間の除外のみを選択した場合には、その名前空間からすべてのメトリクスが除外されます。
   - メトリクスストリームにメトリクス名前空間またはメトリクスをいくつか含めるには、[含める] を選択し、含める名前空間またはメトリクスを選択します。メトリクスを選択せずに名前空間を含めるよう選択した場合には、その名前空間のすべてのメトリクスが含まれます。

9. (オプション) これらのメトリクスの一部について、Minimum (最小値)、Maximum (最大値)、SampleCount (サンプル数)、Sum (合計) 以外の追加の統計情報をストリーミングするには、[統計をさらに追加] を選択します。[Add recommended metrics] (推奨メトリクスを追加する) を選択してよく使用される統計情報を追加するか、名前空間とメトリクス名を手動で選択して追加の統計情報をストリーミングするかのどちらかを行います。次に、ストリーミングする追加の統計情報を選択します。

   別の追加の統計情報のセットをストリーミングする別のメトリクスグループを選択するには、[Add additional statistics] (統計情報の追加) を選択します。各メトリクスには最大 20 個の追加の統計情報を含めることができ、メトリクスストリーム内の最大 100 個のメトリクスに追加の統計情報を含めることができます。

   追加の統計情報をストリーミングすると、より多くの料金が発生します。詳細については、「ストリーミングできる統計情報」を参照してください。

   追加の統計情報の定義については、「CloudWatch 統計定義」を参照してください。

10. (オプション) [メトリクスストリーム名] で新しいメトリクスストリームの名前をカスタマイズします。

11. [メトリクスストリームの作成] を選択します。

Datadog ストリームのデフォルト

クイックパートナーセットアップによる Datadog へのストリームでは、次のデフォルト設定が使用されます。

- 出力形式: OpenTelemetry 0.7.0
- Firehose ストリームのコンテンツエンコーディング: GZIP
- Firehose ストリームのバッファリングオプション: 60 秒間隔、サイズ 4 MB
- Firehose ストリーム再試行オプション: 所要時間 60 秒

クイックパートナーセットアップを使用して Datadog へのメトリクスストリームを作成し、特定のメトリクスをストリーミングする場合、それらのメトリクスにはデフォルトで追加の統計情報が含まれます。追加の統計情報をストリーミングすると、追加料金が発生する場合があります。統計情報とその課金の詳細については、「ストリーミングできる統計情報」を参照してください。

ストリーミングすることを選択した場合にデフォルトで追加の統計情報が含まれるメトリクスを、以下のリストに示します。ストリーミングを開始する前に、これら追加の統計情報を選択解除することもできます。

- **AWS/Lambda** の **Duration**: p50、p80、p95、p99、p99.9
- **AWS/Lambda** の **PostRuntimeExtensionDuration**: p50、p99
- **AWS/S3** の **FirstByteLatency** と **TotalRequestLatency**: p50、p90、p95、p99、p99.9
- **AWS/Polly** の **ResponseLatency** および AWS/ApplicationELB の **TargetResponseTime**: p50、p90、p95、p99
- **AWS/ApiGateway** の **Latency** と **IntegrationLatency**: p90、p95、p99
- **AWS/ELB** の **Latency** と **TargetResponseTime**: p95、p99
- **AWS/AppRunner** の **RequestLatency**: p50、p95、p99
- **AWS/States** の **ActivityTime**、**ExecutionTime**、**LambdaFunctionRunTime**、**LambdaFunctionScheduleTime**、および **ActivityScheduleTime**: p95、p99
- **AWS/MediaLive** の **EncoderBitRate**、**ConfiguredBitRate**、**ConfiguredBitRateAvailable**: p90
- **AWS/AppSync** の **Latency**: p90

Dynatrace ストリームのデフォルト

クイックパートナーセットアップでは、以下のデフォルト設定で Dynatrace へのストリーミングが行われます。

- 出力形式: OpenTelemetry 0.7.0
- Firehose ストリームのコンテンツエンコーディング GZIP
- Firehose ストリームのバッファリングオプション: 60 秒間隔、サイズ 5 MB
- Firehose ストリーム再試行オプション: 所要時間 600 秒

New Relic ストリームのデフォルト

クイックパートナーセットアップでは、以下のデフォルト設定で New Relic へのストリーミングが行われます。

- 出力形式: OpenTelemetry 0.7.0
- Firehose ストリームのコンテンツエンコーディング: GZIP
- Firehose ストリームのバッファリングオプション: 60 秒間隔、サイズ 1 MB
- Firehose ストリーム再試行オプション: 所要時間 60 秒

Splunk Observability Cloud ストリームのデフォルト

クイックパートナーセットアップでは、以下のデフォルト設定で Splunk Observability Cloud へのストリーミングが行われます。

- 出力形式: OpenTelemetry 0.7.0
- Firehose ストリームのコンテンツエンコーディング: GZIP
- Firehose ストリームのバッファリングオプション: 60 秒間隔、サイズ 1 MB
- Firehose ストリーム再試行オプション: 持続時間 300 秒

Sumo Logic ストリームのデフォルト

クイックパートナーセットアップでは、以下のデフォルト設定で Sumo Logic へのストリーミングが行われます。

- 出力形式: OpenTelemetry 0.7.0
- Firehose ストリームのコンテンツエンコーディング: GZIP

- Firehose ストリームのバッファリングオプション: 60 秒間隔、サイズ 1 MB
- Firehose ストリーム再試行オプション: 所要時間 60 秒

ストリーミングできる統計情報

メトリクスストリームには、常に次の統計情報が含まれます。`Minimum`、`Maximum`、`SampleCount`、および `Sum`。メトリクスストリームに次の追加の統計情報を含めることもできます。これは、メトリクスごとの選択になります。これらの統計情報の詳細については、「CloudWatch 統計定義」を参照してください。

- p95 や p99 などのパーセンタイル値 (JSON 形式または OpenTelemetry 形式のストリーム)
- トリミング平均 (JSON 形式のストリームのみ)
- ウィンザー化平均 (JSON 形式のストリームのみ)
- トリミング数 (JSON 形式のストリームのみ)
- トリミング合計 (JSON 形式のストリームの場合のみ)
- パーセンタイルランク (JSON 形式のストリームの場合のみ)
- 四分位間平均 (JSON 形式のストリームのみ)

追加の統計情報をストリーミングすると、追加料金が発生します。特定のメトリクスに対してこれらの追加の統計情報を 1~5 種類ストリーミングすると、追加のメトリクス更新として課金されます。その後、これらの統計情報のうち 5 種類まで追加するごとに、別のメトリクス更新として請求されます。

例えば、あるメトリクスについて、p95、p99、p99.9、トリミング平均、ウィンザー化平均、およびトリミング合計の 6 種類の統計情報をストリーミングするとします。このメトリクスの更新ごとに 3 つのメトリクス更新として課金されます。デフォルトの統計情報を含むメトリクス更新に対して 1 つ、最初の 5 種類の追加の統計情報に対して 1 つ、さらに 6 番目の追加の統計情報に対して 1 つです。さらに 4 種類までの統計情報を追加して合計 10 種類にしても、課金は増加しませんが、11 番目の統計情報を追加すると、課金が増加します。

追加の統計情報をストリーミングするためにメトリクス名と名前空間の組み合わせを指定すると、そのメトリクス名と名前空間のすべてのディメンションの組み合わせが、追加の統計情報とともにストリーミングされます。

CloudWatch メトリクスストリームによって、新しいメトリクス `TotalMetricUpdate` が公開されます。これは、基本のメトリクス更新数と、追加の統計情報のストリーミングによって発生する追加

のメトリクス更新数を反映しています。詳細については、「CloudWatch メトリクスを使用してメトリクスストリームをモニタリングする」を参照してください。

詳細については、「Amazon CloudWatch 料金表」をご覧ください。

> **Note**
>
> 一部のメトリクスはパーセンタイルをサポートしていません。これらのメトリクスのパーセンタイル統計情報はストリームから除外され、メトリクスストリームの料金は発生しません。パーセンタイルをサポートしない統計情報の例としては、AWS/ECS 名前空間の一部のメトリクスがあります。

設定した追加の統計情報は、ストリームのフィルターと一致する場合にのみストリーミングされます。例えば、include フィルターに EC2 と RDS のみを含むストリームを作成し、統計情報設定のリストに EC2 と Lambda が含まれる場合、ストリームには、追加の統計情報を含む EC2 メトリクスと、デフォルトの統計情報のみを含む RDS メトリクスが含まれ、Lambda 統計情報はまったく含まれません。

メトリクスストリームの操作とメンテナンス

メトリクスストリームは、常に、実行中または停止の 2 つの状態のいずれかになります。

- 実行中 – メトリクスストリームは正常に実行されています。ストリーム上のフィルターの設定によっては、送信先にストリーミングされるメトリクスデータがない可能性があります。
- 停止 – メトリクスストリームは、エラーのためではなく、誰かによって明示的に停止されています。ストリームを削除せずにデータのストリーミングを一時的に停止するには、ストリームを停止すると便利です。

メトリクスストリームを停止して再起動すると、メトリクスストリームの停止中に CloudWatch にパブリッシュされたメトリクスデータは、メトリクスストリームにバックフィルされません。

メトリクスストリームの出力形式を変更すると、場合によっては、少量のメトリクスデータが古い形式と新しい形式の両方で送信先に書き込まれることがあります。このような状況を回避するには、現在の配信ストリームと同じ構成で新しい Firehose 配信ストリームを作成し、新しい Firehose 配信ストリームに変更して、出力形式を同時に変更することができます。このようにして、異なる出力形式の Kinesis レコードは、Amazon S3 に別々のオブジェクトとして格納されます。その後、トラフィックを元の Firehose 配信ストリームに戻し、2 番目の配信ストリームを削除できます。

メトリクスストリームを表示、編集、停止、開始するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで、 [Metrics] (メトリクス)、[Streams] (ストリーム) の順に選択します。

   ストリームのリストが表示され、[ステータス] 列には、各ストリームが実行中か停止しているかが表示されます。
3. メトリクスストリームを停止または開始するには、ストリームを選択して [停止] または [開始] をクリックします。
4. メトリクスストリームの詳細を表示するには、ストリームを選択し、[詳細を表示] をクリックします。
5. ストリームの出力形式、フィルタ、送信先 Firehose ストリーム、またはロールを変更するには、[編集] を選択して必要な変更を行います。

   フィルタを変更すると、移行中にメトリクスデータにギャップが生じることがあります。

CloudWatch メトリクスを使用してメトリクスストリームをモニタリングする

メトリクスストリームは、その健全性と AWS/CloudWatch/MetricStreams 名前空間内の操作に関する CloudWatch メトリクスを出力します。以下のメトリクスが出力されます。これらのメトリクスは、MetricStreamName ディメンション付きで出力され、ディメンションなしでも出力されます。ディメンションなしのメトリクスを使用すると、すべてのメトリクスストリームを集計したメトリクスを確認できます。MetricStreamName ディメンション付きのメトリクスを使用すると、そのメトリクスストリームに関するメトリクスだけを表示できます。

これらのメトリクスのすべてについて、値は [Running] (実行中) 状態のメトリクスストリームに対してのみ出力されます。

メトリクス	説明
MetricUpdate	メトリクスストリーミングに送信されたメトリクス更新の数。一定期間中にメトリクスの更新がストリーミングされない場合、このメトリクスはその期間中に出力されません。 メトリクスストリームを停止すると、メトリクスストリームが再び開始されるまで、このメトリクスの出力が停止します。

メトリクス	説明
	有効な統計: Sum 単位: なし
TotalMetricUpdate	これは、次のように計算されます。MetricUpdate + ストリーミングされる追加の統計情報に基づく数値。 一意の名前空間とメトリクス名の組み合わせごとに、追加の統計情報を 1~5 種類ストリーミングすると、TotalMetricUpdate に 1 が加算され、追加の統計情報を 6~10 種類ストリーミングすると、TotalMetricUpdate に 2 が加算され、以下同様となります。 有効な統計: Sum 単位: なし
PublishErrorRate	データを Firehose 配信ストリームに入れるときに発生する、回復不能なエラーの数。一定期間中にエラーが発生しない場合、このメトリクスはその期間中に出力されません。 メトリクスストリームを停止すると、メトリクスストリームが再び開始されるまで、このメトリクスの出力が停止します。 有効な統計: Average メトリクスの更新が書き込めない割合が確認できます。この値は 0.0 から 1.0 の間になります。 単位: なし

CloudWatch と Firehose 間の信頼

Firehose 配信ストリームは、Firehose への書き込みアクセス許可を持つ IAM ロールを通じて CloudWatch を信頼する必要があります。このアクセス許可は、CloudWatch メトリクスストリームが使用する 1 つの Firehose 配信ストリームに制限できます。IAM ロールは、streams.metrics.cloudwatch.amazonaws.com サービスプリンシパルを信頼する必要があります。

CloudWatch コンソールを使用してメトリクスストリームを作成する場合は、CloudWatch で、適切なアクセス許可があるロールを作成できます。別の方法を使用してメトリクスストリームを作成する

場合、または IAM ロール自体を作成する場合は、次のアクセス許可ポリシーと信頼ポリシーを含める必要があります。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "firehose:PutRecord",
                "firehose:PutRecordBatch"
            ],
            "Effect": "Allow",
            "Resource": "arn:aws:firehose:region:account-id:deliverystream/*"
        }
    ]
}
```

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "streams.metrics.cloudwatch.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

メトリクスデータは、メトリクスストリームリソースを所有するソースに代わって、CloudWatch によって送信先の Firehose 配信ストリームにストリーミングされます。

メトリクスストリームの出力形式

CloudWatch メトリクスストリームのデータは、JSON 形式または OpenTelemetry 形式にすることができます。現時点では、OpenTelemetry 1.0.0 形式と 0.7.0 形式がサポートされています。

目次

JSON 形式

JSON 形式を使用する CloudWatch メトリクスストリームでは、各 Firehose レコードには、改行文字 (\n) で区切られた複数の JSON オブジェクトが含まれます。各オブジェクトには、1 つのメトリクスの 1 つのデータポイントが含まれます。

使用される JSON 形式は、AWS Glue および Amazon Athena との完全な互換性があります。Firehose 配信ストリームと AWS Glue テーブルの形式が正しい場合は、S3 に格納する前に、その形式を Parquet 形式または Optimized Row Columnar (ORC) 形式に自動的に変換できます。形式の変換の詳細については、「Converting Your Input Record Format in Firehose」を参照してください。AWS Glue の正しい形式の詳細については、JSON 出力フォーマットにはどの AWS Glue スキーマを使用すればよいですか? を参照してください。

JSON 形式では、`unit` の有効な値は、`MetricDatum` API 構造体の `unit` の値と同じです。詳細については、「MetricDatum」を参照してください。`timestamp` フィールドの値は、`1616004674229` のようなエポックミリ秒単位の値です。

この形式の例を次に示します。この例では、JSON は読みやすいようにフォーマットされていますが、実際にはフォーマット全体が 1 行になっています。

```
{
    "metric_stream_name": "MyMetricStream",
    "account_id": "1234567890",
    "region": "us-east-1",
    "namespace": "AWS/EC2",
    "metric_name": "DiskWriteOps",
    "dimensions": {
        "InstanceId": "i-123456789012"
    },
    "timestamp": 1611929698000,
    "value": {
        "count": 3.0,
```

```
        "sum": 20.0,
        "max": 18.0,
        "min": 0.0,
        "p99": 17.56,
        "p99.9": 17.8764,
        "TM(25%:75%)": 16.43
    },
    "unit": "Seconds"
}
```

JSON 出力フォーマットにはどの AWS Glue スキーマを使用すればよいですか?

AWS Glue テーブルの StorageDescriptor を JSON 表現した例を次に示します。これはその後 Firehose で使用されます。StorageDescriptor の詳細については、「StorageDescriptor」を参照してください。

```
{
  "Columns": [
    {
      "Name": "metric_stream_name",
      "Type": "string"
    },
    {
      "Name": "account_id",
      "Type": "string"
    },
    {
      "Name": "region",
      "Type": "string"
    },
    {
      "Name": "namespace",
      "Type": "string"
    },
    {
      "Name": "metric_name",
      "Type": "string"
    },
    {
      "Name": "timestamp",
      "Type": "timestamp"
    },
    {
```

```
      "Name": "dimensions",
      "Type": "map<string,string>"
    },
    {
      "Name": "value",
      "Type":
 "struct<min:double,max:double,count:double,sum:double,p99:double,p99.9:double>"
    },
    {
      "Name": "unit",
      "Type": "string"
    }
  ],
  "Location": "s3://my-s3-bucket/",
  "InputFormat": "org.apache.hadoop.mapred.TextInputFormat",
  "OutputFormat": "org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat",
  "SerdeInfo": {
    "SerializationLibrary": "org.apache.hive.hcatalog.data.JsonSerDe"
  },
  "Parameters": {
    "classification": "json"
  }
}
```

前の例は、JSON 形式で Amazon S3 に書き込まれたデータの場合です。次のフィールドの値を指定された値に置き換えて、Parquet 形式またはOptimized Row Columnar (ORC) 形式でデータを格納します。

- Parquet:
  - inputFormat: org.apache.hadoop.hive.ql.io.parquet.MapredParquetInputFormat
  - outputFormat: org.apache.hadoop.hive.ql.io.parquet.MapredParquetOutputFormat
  - SerDeInfo.serializationLib: org.apache.hadoop.hive.ql.io.parquet.serde.ParquetHiveSerDe
  - parameters.classification: parquet
- ORC:
  - inputFormat: org.apache.hadoop.hive.ql.io.orc.OrcInputFormat
  - outputFormat: org.apache.hadoop.hive.ql.io.orc.OrcOutputFormat
  - SerDeInfo.serializationLib: org.apache.hadoop.hive.ql.io.orc.OrcSerde
  - parameters.classification: orc

OpenTelemetry 1.0.0 形式

> **Note**
>
> OpenTelemetry 1.0.0 形式では、メトリクス属性は 0.7.0 形式に使用されている `StringKeyValue` タイプではなく `KeyValue` オブジェクトのリストとしてエンコードされます。コンシューマーから見て、0.7.0 形式と 1.0.0 形式との大きな違いはこれだけです。0.7.0 プロトファイルから生成されたパーサーは、1.0.0 形式でエンコードされたメトリクス属性を解析しません。同じことが逆方向にも言えます。1.0.0 プロトファイルから生成されたパーサーは、0.7.0 形式でエンコードされたメトリクス属性を解析しません。

OpenTelemetry は、ツール、API、および SDK を集めたものです。これを使用して、分析用のテレメトリデータ (メトリクス、ログ、トレース) を計測し、生成し、収集し、エクスポートできます。OpenTelemetry は Cloud Native Computing Foundation の一部です。詳細については、「OpenTelemetry」を参照してください。

OpenTelemetry 1.0.0 の完全な仕様については、「Release version 1.0.0」を参照してください。

Kinesis レコードには、1 つ以上の `ExportMetricsServiceRequest` OpenTelemetry データ構造を含めることができます。各データ構造は、レコード長をバイト単位で示す `UnsignedVarInt32` のヘッダーで始まります。各 `ExportMetricsServiceRequest` には、一度に複数のメトリクスのデータを含めることができます。

`ExportMetricsServiceRequest` OpenTelemetry データ構造のメッセージの文字列表現を次に示します。OpenTelemetry では、Google Protocol Buffers バイナリプロトコルを使用してシリアル化しますが、これは人には読み取り不可能です。

```
resource_metrics {
  resource {
    attributes {
      key: "cloud.provider"
      value {
        string_value: "aws"
      }
    }
    attributes {
      key: "cloud.account.id"
      value {
        string_value: "123456789012"
      }
```

```
    }
    attributes {
      key: "cloud.region"
      value {
        string_value: "us-east-1"
      }
    }
    attributes {
      key: "aws.exporter.arn"
      value {
        string_value: "arn:aws:cloudwatch:us-east-1:123456789012:metric-stream/
MyMetricStream"
      }
    }
  }
  scope_metrics {
    metrics {
      name: "amazonaws.com/AWS/DynamoDB/ConsumedReadCapacityUnits"
      unit: "NoneTranslated"
      summary {
        data_points {
          start_time_unix_nano: 60000000000
          time_unix_nano: 120000000000
          count: 1
          sum: 1.0
          quantile_values {
            value: 1.0
          }
          quantile_values {
            quantile: 0.95
            value: 1.0
          }
          quantile_values {
            quantile: 0.99
            value: 1.0
          }
          quantile_values {
            quantile: 1.0
            value: 1.0
          }
          attributes {
            key: "Namespace"
            value {
              string_value: "AWS/DynamoDB"
```

```
            }
          }
          attributes {
            key: "MetricName"
            value {
              string_value: "ConsumedReadCapacityUnits"
            }
          }
          attributes {
            key: "Dimensions"
            value {
              kvlist_value {
                values {
                  key: "TableName"
                  value {
                    string_value: "MyTable"
                  }
                }
              }
            }
          }
        }
        data_points {
          start_time_unix_nano: 70000000000
          time_unix_nano: 130000000000
          count: 2
          sum: 5.0
          quantile_values {
            value: 2.0
          }
          quantile_values {
            quantile: 1.0
            value: 3.0
          }
          attributes {
            key: "Namespace"
            value {
              string_value: "AWS/DynamoDB"
            }
          }
          attributes {
            key: "MetricName"
            value {
              string_value: "ConsumedReadCapacityUnits"
```

```
            }
          }
          attributes {
            key: "Dimensions"
            value {
              kvlist_value {
                values {
                  key: "TableName"
                  value {
                    string_value: "MyTable"
                  }
                }
              }
            }
          }
        }
      }
    }
  }
}
```

OpenTelemetry メトリクスデータをシリアル化するトップレベルオブジェクト

ExportMetricsServiceRequest は、OpenTelemetry エクスポーターペイロードをシリアル化するトップレベルのラッパーです。これには、ResourceMetrics が 1 つ以上含まれています 。

```
message ExportMetricsServiceRequest {
  // An array of ResourceMetrics.
  // For data coming from a single resource this array will typically contain one
  // element. Intermediary nodes (such as OpenTelemetry Collector) that receive
  // data from multiple origins typically batch the data before forwarding further and
  // in that case this array will contain multiple elements.
  repeated opentelemetry.proto.metrics.v1.ResourceMetrics resource_metrics = 1;
}
```

ResourceMetrics は、MetricData オブジェクトを表すトップレベルオブジェクトです。

```
// A collection of ScopeMetrics from a Resource.
message ResourceMetrics {
  reserved 1000;

  // The resource for the metrics in this message.
```

```
  // If this field is not set then no resource info is known.
  opentelemetry.proto.resource.v1.Resource resource = 1;

  // A list of metrics that originate from a resource.
  repeated ScopeMetrics scope_metrics = 2;

  // This schema_url applies to the data in the "resource" field. It does not apply
  // to the data in the "scope_metrics" field which have their own schema_url field.
  string schema_url = 3;
}
```

Resource オブジェクト

Resource オブジェクトは、値のペアを持つオブジェクトで、メトリクスを生成したリソースに関する情報が含まれます。AWS で作成されたメトリクスの場合、データ構造には、EC2 インスタンスや S3 バケットなど、メトリクスに関連するリソースの Amazon リソースネーム (ARN) が含まれます。

Resource オブジェクトには、attributes という属性が含まれています。これには、キーと値のペアが格納されています。

- cloud.account.id にはアカウント ID が含まれています
- cloud.region にはリージョンが含まれています
- aws.exporter.arn にはメトリクスストリーム ARN が含まれています
- cloud.provider は常に aws です。

```
// Resource information.
message Resource {
  // Set of attributes that describe the resource.
  // Attribute keys MUST be unique (it is not allowed to have more than one
  // attribute with the same key).
  repeated opentelemetry.proto.common.v1.KeyValue attributes = 1;

  // dropped_attributes_count is the number of dropped attributes. If the value is 0,
 then
  // no attributes were dropped.
  uint32 dropped_attributes_count = 2;
}
```

ScopeMetrics オブジェクト

scope フィールドには値を入力しません。エクスポートするメトリクスフィールドにのみ入力します。

```
// A collection of Metrics produced by an Scope.
message ScopeMetrics {
  // The instrumentation scope information for the metrics in this message.
  // Semantically when InstrumentationScope isn't set, it is equivalent with
  // an empty instrumentation scope name (unknown).
  opentelemetry.proto.common.v1.InstrumentationScope scope = 1;

  // A list of metrics that originate from an instrumentation library.
  repeated Metric metrics = 2;

  // This schema_url applies to all metrics in the "metrics" field.
  string schema_url = 3;
}
```

Metric オブジェクト

Metric オブジェクトには、いくつかのメタデータのほか、SummaryDataPoint のリストを含む Summary データフィールドが含まれています。

メトリクスストリームの場合、メタデータは次のとおりです。

- name は amazonaws.com/*metric_namespace*/*metric_name* になります。
- description は空白です。
- unit には、メトリクスデータの単位をユニファイドコードの形式 (大文字と小文字が区別される) の計量単位に変換したものが入力されます。詳細については、「OpenTelemetry 1.0.0 形式への変換」および「The Unified Code For Units of Measure」を参照してください。
- type は SUMMARY になります。

```
message Metric {
  reserved 4, 6, 8;

  // name of the metric, including its DNS name prefix. It must be unique.
  string name = 1;

  // description of the metric, which can be used in documentation.
  string description = 2;
```

```
  // unit in which the metric value is reported. Follows the format
  // described by http://unitsofmeasure.org/ucum.html.
  string unit = 3;

  // Data determines the aggregation type (if any) of the metric, what is the
  // reported value type for the data points, as well as the relatationship to
  // the time interval over which they are reported.
  oneof data {
    Gauge gauge = 5;
    Sum sum = 7;
    Histogram histogram = 9;
    ExponentialHistogram exponential_histogram = 10;
    Summary summary = 11;
  }
}

message Summary {
  repeated SummaryDataPoint data_points = 1;
}
```

SummaryDataPoint オブジェクト

SummaryDataPoint オブジェクトには、DoubleSummary メトリクスの時系列内にあるいずれかのデータポイントの値が含まれます。

```
// SummaryDataPoint is a single data point in a timeseries that describes the
// time-varying values of a Summary metric.
message SummaryDataPoint {
  reserved 1;

  // The set of key/value pairs that uniquely identify the timeseries from
  // where this point belongs. The list may be empty (may contain 0 elements).
  // Attribute keys MUST be unique (it is not allowed to have more than one
  // attribute with the same key).
  repeated opentelemetry.proto.common.v1.KeyValue attributes = 7;

  // StartTimeUnixNano is optional but strongly encouraged, see the
  // the detailed comments above Metric.
  //
  // Value is UNIX Epoch time in nanoseconds since 00:00:00 UTC on 1 January
  // 1970.
  fixed64 start_time_unix_nano = 2;
```

```
  // TimeUnixNano is required, see the detailed comments above Metric.
  //
  // Value is UNIX Epoch time in nanoseconds since 00:00:00 UTC on 1 January
  // 1970.
  fixed64 time_unix_nano = 3;

  // count is the number of values in the population. Must be non-negative.
  fixed64 count = 4;

  // sum of the values in the population. If count is zero then this field
  // must be zero.
  //
  // Note: Sum should only be filled out when measuring non-negative discrete
  // events, and is assumed to be monotonic over the values of these events.
  // Negative events *can* be recorded, but sum should not be filled out when
  // doing so.  This is specifically to enforce compatibility w/ OpenMetrics,
  // see: https://github.com/OpenObservability/OpenMetrics/blob/main/specification/
OpenMetrics.md#summary
  double sum = 5;

  // Represents the value at a given quantile of a distribution.
  //
  // To record Min and Max values following conventions are used:
  // - The 1.0 quantile is equivalent to the maximum value observed.
  // - The 0.0 quantile is equivalent to the minimum value observed.
  //
  // See the following issue for more context:
  // https://github.com/open-telemetry/opentelemetry-proto/issues/125
  message ValueAtQuantile {
    // The quantile of a distribution. Must be in the interval
    // [0.0, 1.0].
    double quantile = 1;

    // The value at the given quantile of a distribution.
    //
    // Quantile values must NOT be negative.
    double value = 2;
  }

  // (Optional) list of values at different quantiles of the distribution calculated
  // from the current snapshot. The quantiles must be strictly increasing.
  repeated ValueAtQuantile quantile_values = 6;

  // Flags that apply to this specific data point.  See DataPointFlags
```

```
  // for the available flags and their meaning.
  uint32 flags = 8;
}
```

詳細については、「 OpenTelemetry 1.0.0 形式への変換」を参照してください。

OpenTelemetry 1.0.0 形式への変換

CloudWatch では、CloudWatch データを OpenTelemetry 形式に配置するために、いくつかの変換が実行されます。

名前空間、メトリクス名、ディメンションの変換

これらの属性は、変換によってエンコードされたキーと値のペアになります。

- 1 つの属性にキー `Namespace` があり、その値はメトリクスの名前空間です。
- 1 つの属性にキー `MetricName` があり、その値はメトリクスの名前です。
- 1 つのペアにキー `Dimensions` があり、その値はキーと値のペアからなるネストされたリストです。このリストの各ペアは、CloudWatch メトリクスディメンションにマッピングされます。ペアのキーはディメンションの名前で、その値はディメンションの値です。

Average、Sum、SampleCount、Min、および Max の変換

Summary データポイントを使用すると、CloudWatch で 1 つのデータポイントを使用してこれらの統計情報をすべてエクスポートできます。

- `startTimeUnixNano` には CloudWatch `startTime` が含まれています
- `timeUnixNano` には CloudWatch `endTime` が含まれています
- `sum` には Sum の統計が含まれています
- `count` には SampleCount の統計が含まれています
- `quantile_values` には 2 つの `valueAtQuantile.value` オブジェクトが含まれています
  - `valueAtQuantile.quantile = 0.0` (を含む)`valueAtQuantile.value = Min value`
  - `valueAtQuantile.quantile = 0.99` (を含む)`valueAtQuantile.value = p99 value`
  - `valueAtQuantile.quantile = 0.999` (を含む)`valueAtQuantile.value = p99.9 value`
  - `valueAtQuantile.quantile = 1.0` (を含む)`valueAtQuantile.value = Max value`

メトリクスストリームを消費するリソースは、Average (Sum/SampleCount) の統計で計算できます。

単位の変換

CloudWatch 単位は、次の表に示すように、計量単位のユニファイドコードの形式 (大文字と小文字を区別する) に変換されます。詳細については、「The Unified Code For Units of Measure」を参照してください。

CloudWatch	OpenTelemetry
秒	s
Second または Seconds	s
マイクロ秒	us
Milliseconds	ms
バイト	方法
Kilobytes	kBy
Megabytes	MBy
Gigabytes	GBy
Terabytes	TBy
Bits	bit
Kilobits	kbit
Megabits	MBit
Gigabits	GBit
Terabits	Tbit
割合 (%)	%
カウント	{Count}

CloudWatch	OpenTelemetry
なし	1

単位がスラッシュで結合されている場合は、両方の単位に OpenTelemetry 変換を適用して変換されます。たとえば、Bytes/Second は By/s に変換されます。

OpenTelemetry 1.0.0 メッセージを解析する方法

このセクションでは、OpenTelemetry 1.0.0 の解析を開始する際に役立つ情報を提供します。

まず、言語固有のバインディングを取得する必要があります。これにより、OpenTelemetry 1.0.0 のメッセージを任意の言語で解析できます。

言語固有のバインディングを取得するには

- 手順は、使用する言語によって異なります。
  - Java を使用するには、次の Maven 依存関係を Java プロジェクトに追加します。OpenTelemetry Java >> 0.14.1。
  - 他の言語を使用するには、次の手順を実行します。

    a. 「Generating Your Classes」のリストをチェックして、言語がサポートされていることを確認します。

    b. 「Download Protocol Buffers」の手順に従って、Protobuf コンパイラをインストールします。

    c. 「Release version 1.0.0」で OpenTelemetry 0.7.0 ProtoBuf 定義をダウンロードします。

    d. ダウンロードした OpenTelemetry 0.7.0 ProtoBuf 定義のルートフォルダにいることを確認します。次に、`src` フォルダを作成し、言語固有のバインディングを生成するコマンドを実行します。詳細は、「Generating Your Classes」を参照してください。

    以下は、Javascript バインディングを生成する方法の例です。

```
protoc --proto_path=./ --js_out=import_style=commonjs,binary:src \
opentelemetry/proto/common/v1/common.proto \
opentelemetry/proto/resource/v1/resource.proto \
opentelemetry/proto/metrics/v1/metrics.proto \
opentelemetry/proto/collector/metrics/v1/metrics_service.proto
```

次のセクションでは、前の手順を使用して構築できる言語固有のバインディングの使用例について説明します。

Java

```
package com.example;

import io.opentelemetry.proto.collector.metrics.v1.ExportMetricsServiceRequest;

import java.io.IOException;
import java.io.InputStream;
import java.util.ArrayList;
import java.util.List;

public class MyOpenTelemetryParser {

    public List<ExportMetricsServiceRequest> parse(InputStream inputStream) throws
 IOException {
        List<ExportMetricsServiceRequest> result = new ArrayList<>();

        ExportMetricsServiceRequest request;
        /* A Kinesis record can contain multiple `ExportMetricsServiceRequest`
           records, each of them starting with a header with an
           UnsignedVarInt32 indicating the record length in bytes:
            ------ -------------------------- ------ ----------------------
           |UINT32|ExportMetricsServiceRequest|UINT32|ExportMetricsService...
            ------ -------------------------- ------ ----------------------
         */
        while ((request =
 ExportMetricsServiceRequest.parseDelimitedFrom(inputStream)) != null) {
            // Do whatever we want with the parsed message
            result.add(request);
        }

        return result;
    }
}
```

Javascript

この例では、生成されたバインディングを含むルートフォルダが ./ であるとしています。

関数 parseRecord の data 引数には、次のいずれかの型を指定できます。

- Uint8Array これが最適です
- Buffer ノードの下では最適です
- Array.*number* 8 ビット整数

```
const pb = require('google-protobuf')
const pbMetrics =
    require('./opentelemetry/proto/collector/metrics/v1/metrics_service_pb')

function parseRecord(data) {
    const result = []

    // Loop until we've read all the data from the buffer
    while (data.length) {
        /* A Kinesis record can contain multiple `ExportMetricsServiceRequest`
           records, each of them starting with a header with an
           UnsignedVarInt32 indicating the record length in bytes:
            ------ -------------------------- ------ ----------------------
           |UINT32|ExportMetricsServiceRequest|UINT32|ExportMetricsService...
            ------ -------------------------- ------ ----------------------
         */
        const reader = new pb.BinaryReader(data)
        const messageLength = reader.decoder_.readUnsignedVarint32()
        const messageFrom = reader.decoder_.cursor_
        const messageTo = messageFrom + messageLength

        // Extract the current `ExportMetricsServiceRequest` message to parse
        const message = data.subarray(messageFrom, messageTo)

        // Parse the current message using the ProtoBuf library
        const parsed =
            pbMetrics.ExportMetricsServiceRequest.deserializeBinary(message)

        // Do whatever we want with the parsed message
        result.push(parsed.toObject())

        // Shrink the remaining buffer, removing the already parsed data
        data = data.subarray(messageTo)
    }

    return result
}
```

Python

var-int デリミタは自分で読み取るか、内部メソッド _VarintBytes(size) と _DecodeVarint32(buffer, position) を使用する必要があります。これらは、バッファ内でのサイズバイトの直後の位置を返します。読み取り側は、メッセージからそのバイトのみを読み取る新しいバッファを構築します。

```
size = my_metric.ByteSize()
f.write(_VarintBytes(size))
f.write(my_metric.SerializeToString())
msg_len, new_pos = _DecodeVarint32(buf, 0)
msg_buf = buf[new_pos:new_pos+msg_len]
request = metrics_service_pb.ExportMetricsServiceRequest()
request.ParseFromString(msg_buf)
```

Go

Buffer.DecodeMessage() を使用します。

C#

CodedInputStream を使用します。このクラスは、サイズ区切りのメッセージを読み取ることができます。

C++

google/protobuf/util/delimited_message_util.h に記述されている関数は、サイズ区切りのメッセージを読み取ることができます。

その他の言語

その他の言語については、「Download Protocol Buffers」を参照してください。

パーサーを実装するときは、Kinesis レコードに複数の ExportMetricsServiceRequest Protocol Buffers メッセージを含めることができることを考慮してください。各メッセージは、レコードの長さをバイト単位で示す UnsignedVarInt32 のヘッダーで始まります。

OpenTelemetry 0.7.0 形式

OpenTelemetry は、ツール、API、および SDK を集めたものです。これを使用して、分析用のテレメトリデータ (メトリクス、ログ、トレース) を計測し、生成し、収集し、エクスポートできます。OpenTelemetry は Cloud Native Computing Foundation の一部です。詳細については、「OpenTelemetry」を参照してください。

OpenTelemetry 0.7.0 の完全な仕様については、「v0.7.0 release」を参照してください。

Kinesis レコードには、1 つ以上の ExportMetricsServiceRequest OpenTelemetry データ構造を含めることができます。各データ構造は、レコード長をバイト単位で示す UnsignedVarInt32 のヘッダーで始まります。各 ExportMetricsServiceRequest には、一度に複数のメトリクスのデータを含めることができます。

ExportMetricsServiceRequest OpenTelemetry データ構造のメッセージの文字列表現を次に示します。OpenTelemetry では、Google Protocol Buffers バイナリプロトコルを使用してシリアル化しますが、これは人には読み取り不可能です。

```
resource_metrics {
  resource {
    attributes {
      key: "cloud.provider"
      value {
        string_value: "aws"
      }
    }
    attributes {
      key: "cloud.account.id"
      value {
        string_value: "2345678901"
      }
    }
    attributes {
      key: "cloud.region"
      value {
        string_value: "us-east-1"
      }
    }
    attributes {
      key: "aws.exporter.arn"
      value {
        string_value: "arn:aws:cloudwatch:us-east-1:123456789012:metric-stream/
MyMetricStream"
      }
    }
  }
  instrumentation_library_metrics {
    metrics {
      name: "amazonaws.com/AWS/DynamoDB/ConsumedReadCapacityUnits"
      unit: "1"
```

```
      double_summary {
        data_points {
          labels {
            key: "Namespace"
            value: "AWS/DynamoDB"
          }
          labels {
            key: "MetricName"
            value: "ConsumedReadCapacityUnits"
          }
          labels {
            key: "TableName"
            value: "MyTable"
          }
          start_time_unix_nano: 1604948400000000000
          time_unix_nano: 1604948460000000000
          count: 1
          sum: 1.0
          quantile_values {
            quantile: 0.0
            value: 1.0
          }
          quantile_values {
            quantile: 0.95
            value: 1.0
          }
          quantile_values {
            quantile: 0.99
            value: 1.0
          }
          quantile_values {
            quantile: 1.0
            value: 1.0
          }
        }
        data_points {
          labels {
            key: "Namespace"
            value: "AWS/DynamoDB"
          }
          labels {
            key: "MetricName"
            value: "ConsumedReadCapacityUnits"
          }
```

```
          labels {
            key: "TableName"
            value: "MyTable"
          }
          start_time_unix_nano: 1604948460000000000
          time_unix_nano: 1604948520000000000
          count: 2
          sum: 5.0
          quantile_values {
            quantile: 0.0
            value: 2.0
          }
          quantile_values {
            quantile: 1.0
            value: 3.0
          }
        }
      }
    }
  }
}
```

OpenTelemetry メトリクスデータをシリアル化するトップレベルオブジェクト

ExportMetricsServiceRequest は、OpenTelemetry エクスポーターペイロードをシリアル化するトップレベルのラッパーです。これには、ResourceMetrics が 1 つ以上含まれています 。

```
message ExportMetricsServiceRequest {
  // An array of ResourceMetrics.
  // For data coming from a single resource this array will typically contain one
  // element. Intermediary nodes (such as OpenTelemetry Collector) that receive
  // data from multiple origins typically batch the data before forwarding further and
  // in that case this array will contain multiple elements.
  repeated opentelemetry.proto.metrics.v1.ResourceMetrics resource_metrics = 1;
}
```

ResourceMetrics は、MetricData オブジェクトを表すトップレベルオブジェクトです。

```
// A collection of InstrumentationLibraryMetrics from a Resource.
message ResourceMetrics {
  // The resource for the metrics in this message.
  // If this field is not set then no resource info is known.
  opentelemetry.proto.resource.v1.Resource resource = 1;
```

```
  // A list of metrics that originate from a resource.
  repeated InstrumentationLibraryMetrics instrumentation_library_metrics = 2;
}
```

Resource オブジェクト

Resource オブジェクトは、値のペアを持つオブジェクトで、メトリクスを生成したリソースに関する情報が含まれます。AWS で作成されたメトリクスの場合、データ構造には、EC2 インスタンスや S3 バケットなど、メトリクスに関連するリソースの Amazon リソースネーム (ARN) が含まれます。

Resource オブジェクトには、attributes という属性が含まれています。これには、キーと値のペアが格納されています。

- cloud.account.id にはアカウント ID が含まれています
- cloud.region にはリージョンが含まれています
- aws.exporter.arn にはメトリクスストリーム ARN が含まれています
- cloud.provider は常に aws です。

```
// Resource information.
message Resource {
  // Set of labels that describe the resource.
  repeated opentelemetry.proto.common.v1.KeyValue attributes = 1;

  // dropped_attributes_count is the number of dropped attributes. If the value is 0,
  // no attributes were dropped.
  uint32 dropped_attributes_count = 2;
}
```

InstrumentationLibraryMetrics オブジェクト

instrumentation_library フィールドには入力されません。エクスポートするメトリクスフィールドのみに入力されます。

```
// A collection of Metrics produced by an InstrumentationLibrary.
message InstrumentationLibraryMetrics {
  // The instrumentation library information for the metrics in this message.
  // If this field is not set then no library info is known.
```

```
  opentelemetry.proto.common.v1.InstrumentationLibrary instrumentation_library = 1;
  // A list of metrics that originate from an instrumentation library.
  repeated Metric metrics = 2;
}
```

Metric オブジェクト

Metric オブジェクトには、DoubleSummaryDataPoint のリストを含む DoubleSummary データフィールドが含まれています。

```
message Metric {
  // name of the metric, including its DNS name prefix. It must be unique.
  string name = 1;

  // description of the metric, which can be used in documentation.
  string description = 2;

  // unit in which the metric value is reported. Follows the format
  // described by http://unitsofmeasure.org/ucum.html.
  string unit = 3;

  oneof data {
    IntGauge int_gauge = 4;
    DoubleGauge double_gauge = 5;
    IntSum int_sum = 6;
    DoubleSum double_sum = 7;
    IntHistogram int_histogram = 8;
    DoubleHistogram double_histogram = 9;
    DoubleSummary double_summary = 11;
  }
}

message DoubleSummary {
  repeated DoubleSummaryDataPoint data_points = 1;
}
```

MetricDescriptor オブジェクト

MetricDescriptor オブジェクトには、メタデータが含まれています。詳細については、GitHub の「metrics.proto」を参照してください。

メトリクスストリームの場合、MetricDescriptor には次の内容があります。

- name は amazonaws.com/*metric_namespace*/*metric_name* になります。
- description は空白になります。
- unit には、メトリクスデータの単位をユニファイドコードの形式 (大文字と小文字が区別される) の計量単位に変換したものが入力されます。詳細については、「 OpenTelemetry 0.7.0 形式への変換」および「The Unified Code For Units of Measure」を参照してください。
- type は SUMMARY になります。

DoubleSummaryDataPoint オブジェクト

DoubleSumMaryDataPointオブジェクトには、DoubleSumMaryメトリクスの時系列内の 1 つのデータポイントの値が含まれます。

```
// DoubleSummaryDataPoint is a single data point in a timeseries that describes the
// time-varying values of a Summary metric.
message DoubleSummaryDataPoint {
  // The set of labels that uniquely identify this timeseries.
  repeated opentelemetry.proto.common.v1.StringKeyValue labels = 1;

  // start_time_unix_nano is the last time when the aggregation value was reset
  // to "zero". For some metric types this is ignored, see data types for more
  // details.
  //
  // The aggregation value is over the time interval (start_time_unix_nano,
  // time_unix_nano].
  //
  // Value is UNIX Epoch time in nanoseconds since 00:00:00 UTC on 1 January
  // 1970.
  //
  // Value of 0 indicates that the timestamp is unspecified. In that case the
  // timestamp may be decided by the backend.
  fixed64 start_time_unix_nano = 2;

  // time_unix_nano is the moment when this aggregation value was reported.
  //
  // Value is UNIX Epoch time in nanoseconds since 00:00:00 UTC on 1 January
  // 1970.
  fixed64 time_unix_nano = 3;

  // count is the number of values in the population. Must be non-negative.
  fixed64 count = 4;
```

```
  // sum of the values in the population. If count is zero then this field
  // must be zero.
  double sum = 5;

  // Represents the value at a given quantile of a distribution.
  //
  // To record Min and Max values following conventions are used:
  // - The 1.0 quantile is equivalent to the maximum value observed.
  // - The 0.0 quantile is equivalent to the minimum value observed.
  message ValueAtQuantile {
    // The quantile of a distribution. Must be in the interval
    // [0.0, 1.0].
    double quantile = 1;

    // The value at the given quantile of a distribution.
    double value = 2;
  }

  // (Optional) list of values at different quantiles of the distribution calculated
  // from the current snapshot. The quantiles must be strictly increasing.
  repeated ValueAtQuantile quantile_values = 6;
}
```

詳細については、「 OpenTelemetry 0.7.0 形式への変換」を参照してください。

OpenTelemetry 0.7.0 形式への変換

CloudWatch では、CloudWatch データを OpenTelemetry 形式に配置するために、いくつかの変換が実行されます。

名前空間、メトリクス名、ディメンションの変換

これらの属性は、変換によってエンコードされたキーと値のペアになります。

- メトリクスの名前空間が含まれるペアが 1 つあります
- メトリクスの名前が含まれるペアが 1 つあります
- CloudWatch は、ディメンションごとに、`metricDatum.Dimensions[i].`*`Name`*`, metricDatum.Dimensions[i].`*`Value`* のペアを保存します。

Average、Sum、SampleCount、Min、および Max の変換

Summary データポイントを使用すると、CloudWatch で 1 つのデータポイントを使用してこれらの統計情報をすべてエクスポートできます。

- `startTimeUnixNano` には CloudWatch `startTime` が含まれています
- `timeUnixNano` には CloudWatch `endTime` が含まれています
- `sum` には Sum の統計が含まれています
- `count` には SampleCount の統計が含まれています
- `quantile_values` には 2 つの `valueAtQuantile.value` オブジェクトが含まれています
  - `valueAtQuantile.quantile = 0.0` (を含む)`valueAtQuantile.value = Min value`
  - `valueAtQuantile.quantile = 0.99` (を含む)`valueAtQuantile.value = p99 value`
  - `valueAtQuantile.quantile = 0.999` (を含む)`valueAtQuantile.value = p99.9 value`
  - `valueAtQuantile.quantile = 1.0` (を含む)`valueAtQuantile.value = Max value`

メトリクスストリームを消費するリソースは、Average (Sum/SampleCount) の統計で計算できます。

単位の変換

CloudWatch 単位は、次の表に示すように、計量単位のユニファイドコードの形式 (大文字と小文字を区別する) に変換されます。詳細については、「The Unified Code For Units of Measure」を参照してください。

CloudWatch	OpenTelemetry
秒	s
Second または Seconds	s
Microsecond	us
Milliseconds	ms
バイト	方法
Kilobytes	kBy

CloudWatch	OpenTelemetry
Megabytes	MBy
Gigabytes	GBy
Terabytes	TBy
Bits	bit
Kilobits	kbit
Megabits	MBit
Gigabits	GBit
Terabits	Tbit
割合 (%)	%
カウント	{Count}
なし	1

単位がスラッシュで結合されている場合は、両方の単位に OpenTelemetry 変換を適用して変換されます。たとえば、Bytes/Second は By/s に変換されます。

OpenTelemetry 0.7.0 メッセージを解析する方法

このセクションでは、OpenTelemetry 0.7.0 の解析を開始する際に役立つ情報を提供します。

まず、言語固有のバインディングを取得する必要があります。これにより、OpenTelemetry 0.7.0 のメッセージを適宜の言語で解析できます。

言語固有のバインディングを取得するには

- 手順は、使用する言語によって異なります。
  - Java を使用するには、次の Maven 依存関係を Java プロジェクトに追加します。OpenTelemetry Java >> 0.14.1。
  - 他の言語を使用するには、次の手順を実行します。

a. 「Generating Your Classes」のリストをチェックして、言語がサポートされていることを確認します。

b. 「Download Protocol Buffers」の手順に従って、Protobuf コンパイラをインストールします。

c. 「v0.7.0 release」で OpenTelemetry 0.7.0 ProtoBuf 定義をダウンロードします。

d. ダウンロードした OpenTelemetry 0.7.0 ProtoBuf 定義のルートフォルダにいることを確認します。次に、`src` フォルダを作成し、言語固有のバインディングを生成するコマンドを実行します。詳細は、「Generating Your Classes」を参照してください。

   以下は、Javascript バインディングを生成する方法の例です。

```
protoc --proto_path=./ --js_out=import_style=commonjs,binary:src \
opentelemetry/proto/common/v1/common.proto \
opentelemetry/proto/resource/v1/resource.proto \
opentelemetry/proto/metrics/v1/metrics.proto \
opentelemetry/proto/collector/metrics/v1/metrics_service.proto
```

次のセクションでは、前の手順を使用して構築できる言語固有のバインディングの使用例について説明します。

Java

```
package com.example;

import io.opentelemetry.proto.collector.metrics.v1.ExportMetricsServiceRequest;

import java.io.IOException;
import java.io.InputStream;
import java.util.ArrayList;
import java.util.List;

public class MyOpenTelemetryParser {

    public List<ExportMetricsServiceRequest> parse(InputStream inputStream) throws
 IOException {
        List<ExportMetricsServiceRequest> result = new ArrayList<>();

        ExportMetricsServiceRequest request;
        /* A Kinesis record can contain multiple `ExportMetricsServiceRequest`
```

```
            records, each of them starting with a header with an
            UnsignedVarInt32 indicating the record length in bytes:
             ------ -------------------------- ------ ----------------------
            |UINT32|ExportMetricsServiceRequest|UINT32|ExportMetricsService...
             ------ -------------------------- ------ ----------------------
         */
        while ((request =
 ExportMetricsServiceRequest.parseDelimitedFrom(inputStream)) != null) {
            // Do whatever we want with the parsed message
            result.add(request);
        }

        return result;
    }
}
```

Javascript

この例では、生成されたバインディングを含むルートフォルダが ./ であるとしています。

関数 parseRecord の data 引数には、次のいずれかの型を指定できます。

- Uint8Array これが最適です
- Buffer ノードの下では最適です
- Array.*number* 8 ビット整数

```
const pb = require('google-protobuf')
const pbMetrics =
    require('./opentelemetry/proto/collector/metrics/v1/metrics_service_pb')

function parseRecord(data) {
    const result = []

    // Loop until we've read all the data from the buffer
    while (data.length) {
        /* A Kinesis record can contain multiple `ExportMetricsServiceRequest`
            records, each of them starting with a header with an
            UnsignedVarInt32 indicating the record length in bytes:
             ------ -------------------------- ------ ----------------------
            |UINT32|ExportMetricsServiceRequest|UINT32|ExportMetricsService...
             ------ -------------------------- ------ ----------------------
         */
```

```
        const reader = new pb.BinaryReader(data)
        const messageLength = reader.decoder_.readUnsignedVarint32()
        const messageFrom = reader.decoder_.cursor_
        const messageTo = messageFrom + messageLength

        // Extract the current `ExportMetricsServiceRequest` message to parse
        const message = data.subarray(messageFrom, messageTo)

        // Parse the current message using the ProtoBuf library
        const parsed =
            pbMetrics.ExportMetricsServiceRequest.deserializeBinary(message)

        // Do whatever we want with the parsed message
        result.push(parsed.toObject())

        // Shrink the remaining buffer, removing the already parsed data
        data = data.subarray(messageTo)
    }

    return result
}
```

Python

var-int デリミタは自分で読み取るか、内部メソッド _VarintBytes(size) と _DecodeVarint32(buffer, position) を使用する必要があります。これらは、バッファ内でのサイズバイトの直後の位置を返します。読み取り側は、メッセージからそのバイトのみを読み取る新しいバッファを構築します。

```
size = my_metric.ByteSize()
f.write(_VarintBytes(size))
f.write(my_metric.SerializeToString())
msg_len, new_pos = _DecodeVarint32(buf, 0)
msg_buf = buf[new_pos:new_pos+msg_len]
request = metrics_service_pb.ExportMetricsServiceRequest()
request.ParseFromString(msg_buf)
```

Go

Buffer.DecodeMessage() を使用します。

C#

CodedInputStream を使用します。このクラスは、サイズ区切りのメッセージを読み取ることができます。

C++

google/protobuf/util/delimited_message_util.h に記述されている関数は、サイズ区切りのメッセージを読み取ることができます。

その他の言語

その他の言語については、「Download Protocol Buffers」を参照してください。

パーサーを実装するときは、Kinesis レコードに複数の ExportMetricsServiceRequest Protocol Buffers メッセージを含めることができることを考慮してください。各メッセージは、レコードの長さをバイト単位で示す UnsignedVarInt32 のヘッダーで始まります。

トラブルシューティング

最終送信先にメトリクスデータが表示されない場合は、次の点を確認してください。

- メトリクスストリームが実行中状態であることを確認します。CloudWatch コンソールを使用してこれを行う手順については、「メトリクスストリームの操作とメンテナンス」を参照してください。
- 過去 3 日以上公開されたメトリクスはストリーミングされません。メトリクスがストリーミングされるかどうかを判断するには、CloudWatch コンソールでメトリクスをグラフ化して、表示される最新のデータポイントの時間を確認します。3 日以上経過している場合、メトリックスストリームで取得されません。
- メトリクスストリームが出力するメトリクスを確認します。CloudWatch コンソールの [Metrics] (メトリクス) で、[MetricUpdate]、[TotalMetricUpdate]、および [PublishErrorRate] メトリクスの [AWS/CloudWatch/MetricStreams] 名前空間を確認します。
- PublishErrorRate メトリクスが高い場合は、Firehose 配信ストリームで使用される送信先が存在すること、およびメトリクスストリームの設定で指定された IAM ロールが CloudWatch サービスプリンシパルに書き込み許可を与えていることを確認します。詳細については、「CloudWatch と Firehose 間の信頼」を参照してください。
- Firehose 配信ストリームに最終送信先への書き込み許可があることを確認します。
- Firehose コンソールで、メトリクスストリームに使用される Firehose 配信ストリームを表示し、[モニタリング] タブをチェックして、Firehose 配信ストリームがデータを受信しているかどうかを確認します。

- Firehose 配信ストリームに正しい内容が設定されていることを確認します。
- Firehose 配信ストリームが書き込む最終送信先について、ログまたはメトリクスがあればチェックします。
- 詳細情報を取得するには、Firehose 配信ストリームで CloudWatch Logs エラーログを有効にします。詳細については、「Monitoring Amazon Data Firehose Using CloudWatch Logs」を参照してください。

利用可能なメトリクスを表示する

メトリクスはまず名前空間ごとにグループ化され、次に各名前空間内の種々のディメンションの組み合わせごとにグループ化されます。例えば、すべての EC2 メトリクス、インスタンス別にまとめた EC2 メトリクス、Auto Scaling グループ別にまとめた EC2 メトリクスが表示できます。

Amazon CloudWatch にメトリクスを送信するのは、使用中の AWS サービスのみです。

CloudWatch にメトリクスを送信する AWS サービスのリストについては、CloudWatch メトリクスを発行する AWS のサービス を参照してください。このページでは、各サービスによって公開されるメトリクスとディメンションも確認できます。

> **Note**
>
> 過去 2 週間に新しいデータポイントがないメトリクスは、コンソールに表示されません。また、これらはコンソールの [すべてのメトリクス] タブの検索ボックスにメトリクス名やディメンション名を入力しても表示されず、list-metrics コマンドの結果でも返されません。これらのメトリクスを取得する最善の方法は、AWS CLI の get-metric-data コマンドまたは get-metric-statistics コマンドを使用することです。
> 表示したい古いメトリクスとディメンションが類似する現在のメトリクスがある場合は、この現在の類似するメトリクスを表示できます。次に、[ソース] タブをクリックし、メトリクス名とディメンションのフィールドを必要な内容に変更して、時間範囲もメトリクスの報告時間に変更します。

次の手順は、メトリクスの名前空間を参照してメトリクスを検索および表示するのに役立ちます。ターゲットを絞った検索用語を使用してメトリクスを検索することもできます。詳細については、「利用可能なメトリクスを検索する」を参照してください。

CloudWatch クロスアカウントオブザーバビリティでモニターリングアカウントとして設定されたアカウントを参照している場合、このモニターリングアカウントにリンクされているソースアカウント

のメトリクスを表示できます。ソースアカウントのメトリクスが表示されると、そのアカウントの ID またはラベルも表示されます。詳細については、「CloudWatch のクロスアカウントオブザーバビリティ」を参照してください。

コンソールを使い、利用可能なメトリクスを名前空間とディメンション別に表示する

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで、[Metrics]、[All metrics] を選択します。
3. メトリクスの名前空間 ([EC2]、[Lambda] など) を選択します。
4. メトリクスのディメンション ([Per-Instance Metrics] (インスタンス別メトリクス)、[By Function Name] (関数名別) など) を選択します。
5. [Browse] (参照) タブには、名前空間内のそのディメンションのメトリクスがすべて表示されます。それぞれのメトリクス名にある情報ボタンを選択すると、メトリクスの定義をポップアップで確認できます。

   これが CloudWatch クロスアカウントオブザーバビリティのモニターリングアカウントの場合、このモニターリングアカウントにリンクされているソースアカウントのメトリクスも表示されます。表の [Account label]」 (アカウントラベル) 列と [Account id] (アカウント ID) 列には、各指標がどのアカウントからのものかが表示されます。

   以下の操作を行うことができます。

   a. テーブルを並べ替えるには、列見出しを使用します。
   b. メトリクスをグラフ表示するには、メトリクスの横にあるチェックボックスを選択します。すべてのメトリクスを選択するには、テーブルの見出し行にあるチェックボックスを選択します。
   c. アカウント別にフィルタリングするには、アカウントラベルまたはアカウント ID を選択し、[Add to search] (検索に追加) を選択します。
   d. リソースでフィルタするには、リソース ID を選択し、[Add to search] を選択します。
   e. メトリクスでフィルタするには、メトリクスの名前を選択し、[Add to search] を選択します。
6. (オプション) このグラフを CloudWatch ダッシュボードに追加するには、[Actions] (アクション)、[Add to dashboard] (ダッシュボードに追加) の順に選択します。

AWS CLI を使用して、利用可能なメトリクスをアカウントの名前空間、ディメンション、またはメトリクスごとに表示するには

CloudWatch メトリクスを一覧表示するには、list-metrics コマンドを使用します。メトリクスを発行するすべてのサービスの名前空間、メトリクス、ディメンションのリストについては、「CloudWatch メトリクスを発行する AWS のサービス」を参照してください。

次のコマンド例では、Amazon EC2 のすべてのメトリクスを一覧表示します。

```
aws cloudwatch list-metrics --namespace AWS/EC2
```

出力例を次に示します。

```
{
  "Metrics" : [
    ...
    {
        "Namespace": "AWS/EC2",
        "Dimensions": [
            {
                "Name": "InstanceId",
                "Value": "i-1234567890abcdef0"
            }
        ],
        "MetricName": "NetworkOut"
    },
    {
        "Namespace": "AWS/EC2",
        "Dimensions": [
            {
                "Name": "InstanceId",
                "Value": "i-1234567890abcdef0"
            }
        ],
        "MetricName": "CPUUtilization"
    },
    {
        "Namespace": "AWS/EC2",
        "Dimensions": [
            {
                "Name": "InstanceId",
                "Value": "i-1234567890abcdef0"
            }
        ],
        "MetricName": "NetworkIn"
    },
```

```
    ...
  ]
}
```

指定したリソースで利用可能なメトリクスをすべてリストするには

次の例では、指定のインスタンスの結果だけを表示する目的で AWS/EC2 名前空間と InstanceId ディメンションを指定します。

```
aws cloudwatch list-metrics --namespace AWS/EC2 --dimensions
 Name=InstanceId,Value=i-1234567890abcdef0
```

すべてのリソースのメトリクスをリストするには

次の例では、指定のメトリクスの結果だけを表示する目的で AWS/EC2 名前空間とメトリクス名を指定します。

```
aws cloudwatch list-metrics --namespace AWS/EC2 --metric-name CPUUtilization
```

CloudWatch クロスアカウントオブザーバビリティでリンクされたソースアカウントからメトリクスを取得するには

次の例では、モニターリングアカウントで実行され、モニターリングアカウントおよびリンクされたすべてのソースアカウントの両方からメトリクスを取得します。--include-linked-accounts を追加しないと、コマンドはモニターリングアカウントのメトリクスのみを返します。

```
aws cloudwatch list-metrics --include-linked-accounts
```

CloudWatch クロスアカウントオブザーバビリティでソースアカウントからメトリクスを取得するには

次の例では、モニターリングアカウントで実行され、ID 111122223333 のソースアカウントからメトリクスを取得します。

```
aws cloudwatch list-metrics --include-linked-accounts --owning-account "111122223333"
```

利用可能なメトリクスを検索する

ターゲット検索用語を使用して、アカウントにあるすべてのメトリクス中から検索できます。名前空間、メトリクス名、またはディメンション内に一致する結果があるメトリクスが返されます。

これが CloudWatch クロスアカウントオブザーバビリティのモニターリングアカウントの場合は、このモニターリングアカウントにリンクされているソースアカウントのメトリクスも検索します。

> **Note**
>
> 過去 2 週間に新しいデータポイントがないメトリクスは、コンソールに表示されません。また、これらはコンソールの [すべてのメトリクス] タブの検索ボックスにメトリクス名やディメンション名を入力しても表示されず、list-metrics コマンドの結果でも返されません。これらのメトリクスを取得する最善の方法は、AWS CLI の get-metric-data コマンドまたは get-metric-statistics コマンドを使用することです。

CloudWatch で利用可能なメトリクスを検索するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで [Metrics (メトリクス)] を選択します。
3. [All metrics] (すべてのメトリクス) タブ上の検索フィールドに、検索語 (メトリクス名、名前空間、アカウント ID、アカウントラベル、ディメンションの名前か値、リソース名など) を入力します。メトリクスにその検索語が含まれている名前空間がすべて表示されます。

   たとえば、**`volume`** を検索した場合、名前の一部がその検索語であるメトリクスを含む名前空間が表示されます。

   検索の詳細については、「グラフで検索式を使用する」を参照してください。
4. すべての検索結果をグラフ化するには、[グラフの検索] を選択します。

   または

   名前空間を選択して、その名前空間のメトリクスを表示します。続いて、以下のように行います。

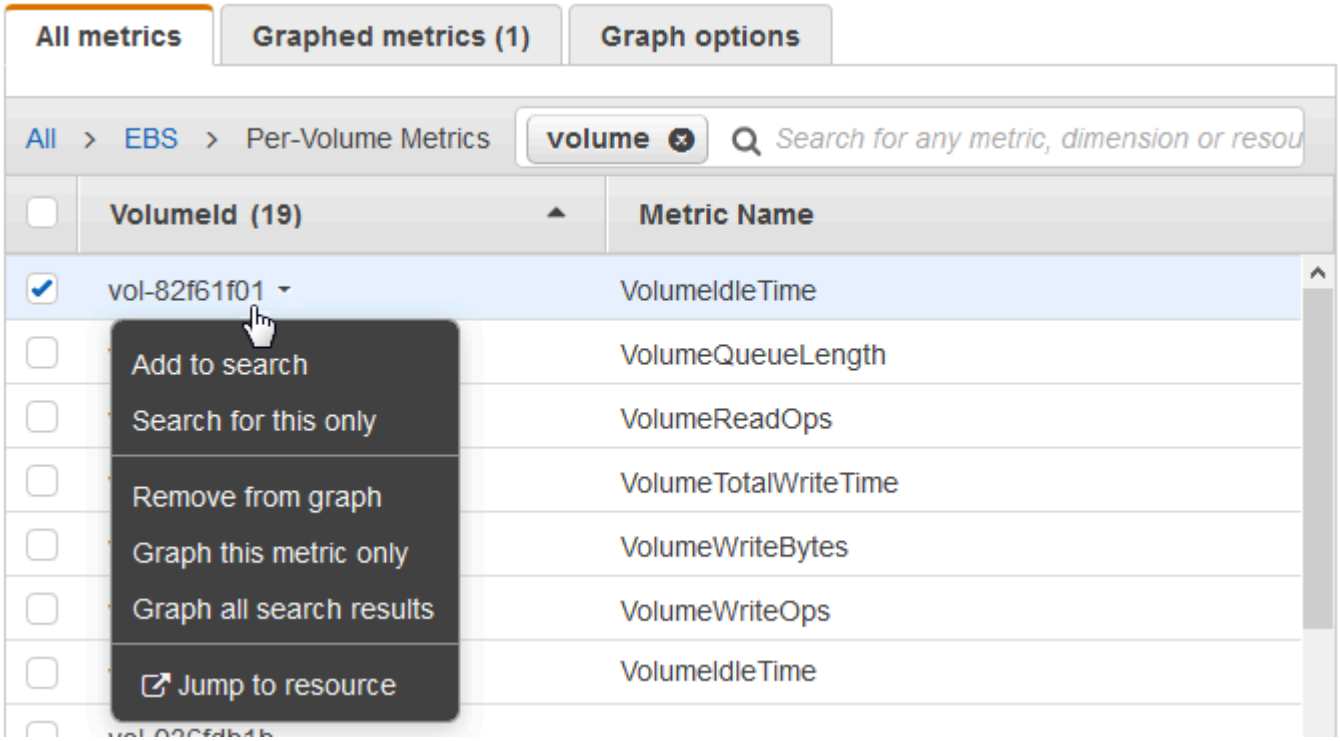
   a. 1 つ以上のメトリクスをグラフ化するには、各メトリクスの横にあるチェックボックスを選択します。すべてのメトリクスを選択するには、テーブルの見出し行にあるチェックボックスを選択します。

   b. 検索を絞り込むには、メトリクス名にカーソルを合わせ、[検索に追加] または [これのみを検索] を選択します。

   c. コンソール内のリソースの 1 つを表示するには、リソース ID を選択し、[リソースへジャンプ] を選択します。

d. メトリクスのヘルプを表示するには、メトリクス名を選択し、[What is this?] (これは何ですか？) を選択します。

選択したメトリクスがグラフに表示されます。



5. (オプション) 検索バーのボタンのいずれかを選択して、検索語のその部分を編集します。

メトリクスのグラフ化

CloudWatch コンソールを使用して、他の AWS のサービスによって生成されたメトリクスデータをグラフ化します。これにより、各サービスのメトリクスアクティビティをより効率的に確認できます。次の手順では、CloudWatch でメトリクスをグラフ化する方法を示します。

内容

- グラフのメトリクスからアラームを作成する

メトリクスをグラフ化する

CloudWatch コンソールを使用してメトリクスを選択し、そのメトリクスデータのグラフを作成できます。

CloudWatch では、メトリクスの次の統計が使用できます: `Average`、`Minimum`、`Maximum`、`Sum`、`SampleCount`。詳細については、「統計」を参照してください。

さまざまな詳細レベルでデータを表示できます。たとえば、1 分間の表示を選択できます。これは、トラブルシューティング時に役立ちます。または、詳細度がより低い 1 時間表示を選択します。これは、時間の経過に伴う傾向を確認できるように、より広い期間 (3 日間など) を表示するときに便利です。詳細については、「期間」を参照してください。

CloudWatch クロスアカウントオブザーバビリティでモニターリングアカウントとして設定されたアカウントを使用している場合、このモニターリングアカウントにリンクされているソースアカウントのメトリクスをグラフ化できます。詳細については、「CloudWatch のクロスアカウントオブザーバビリティ」を参照してください。

グラフの作成

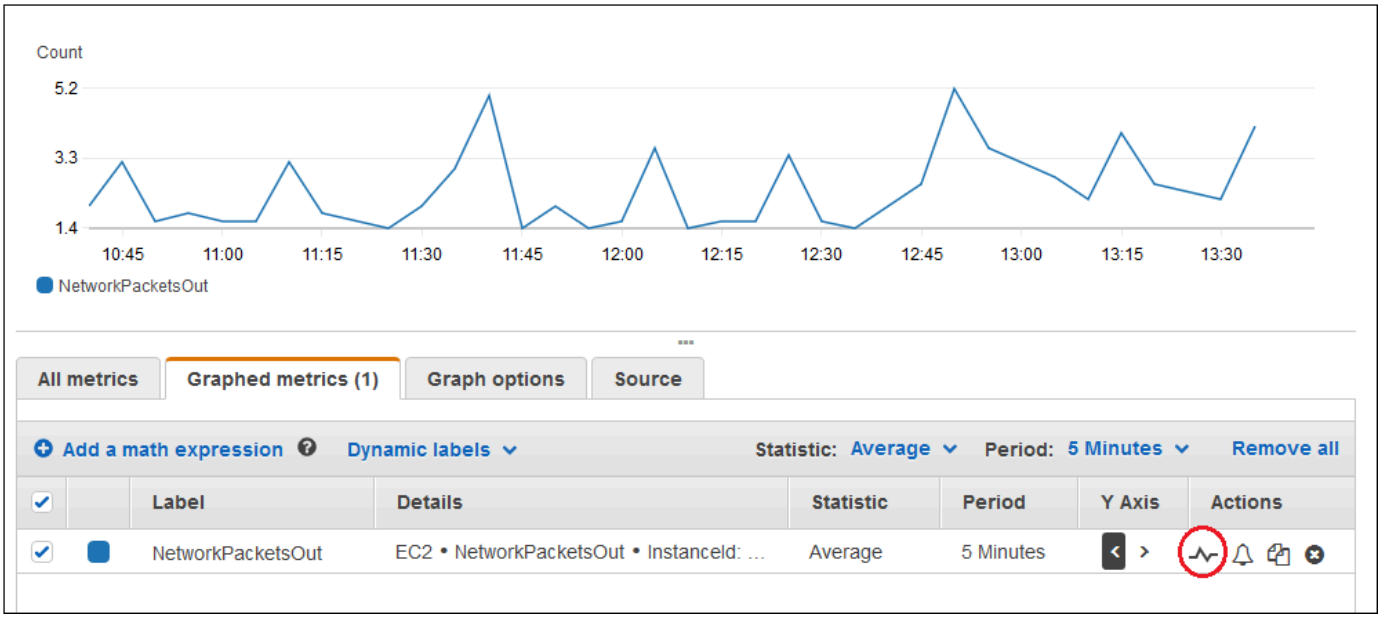
メトリクスをグラフ化するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで、[Metrics]、[All metrics] を選択します。
3. [参照] タブで、検索フィールドに検索語 (メトリクス名、アカウント ID、リソース名など) を入力します。

   たとえば、`CPUUtilization` メトリクスを検索した場合、そのメトリクスを持つ名前空間とディメンションが見つかります。
4. 検索結果の 1 つを選択すると、メトリクスが表示されます。
5. 1 つ以上のメトリクスをグラフ化するには、各メトリクスの横にあるチェックボックスを選択します。すべてのメトリクスを選択するには、テーブルの見出し行にあるチェックボックスを選択します。
6. (オプション) グラフのタイプを変更するには、[オプション] タブを選択します。その後、折れ線グラフ、積み上げ面グラフ、数値表示、ゲージ、棒グラフ、または円グラフから選択できます。

7. [グラフ化したメトリクス] タブを選択します。
8. (オプション) グラフで使用される統計情報を変更するには、メトリクス名の横にある [Statistic] (統計) 列で新しい統計を選択します。

   CloudWatch 統計の詳細については、「CloudWatch 統計定義」を参照してください。pxx パーセンタイル統計の詳細については、「パーセンタイル」を参照してください。
9. (オプション) メトリクスの想定値を示す異常検出バンドを追加するには、メトリクスの横にある [アクション] で異常検出アイコンを選択します。



CloudWatch は、メトリクスの最近の履歴データから最大 2 週間分を使用して、想定値のモデルを計算します。次に、この期待値の範囲をグラフ上のバンドとして表示します。CloudWatch は、メトリクスの下に新しい行を追加して、ANOMALY_DETECTION_BAND と名前が付けられた異常検出バンドの数式を表示します。最近の履歴データが存在する場合は、異常検出バンドのプレビューがすぐに表示されます。これは、モデルによって生成された異常検出バンドの近似値です。実際の異常検出バンドが表示されるまでに最大 15 分かかります。

デフォルトでは、CloudWatch は、想定値のバンドの上限と下限を作成し、バンドのしきい値のデフォルト値を 2 に設定します。この数値を変更するには、バンドの [詳細] の下にある式の末尾の値を変更します。

- (オプション) [Edit model (モデルの編集)] を選択して、異常検出モデルの計算方法を変更します。過去および将来の期間は、モデルの計算のためのトレーニングで使用されないように除外できます。システムの停止、デプロイ、休日などの異常なイベントシステムはトレーニ

ングデータから除外することが重要です。夏時間の変更に合わせてモデルで使用するタイムゾーンを指定することもできます。

詳細については、「CloudWatch 異常検出の使用」を参照してください。

グラフでモデルを非表示にするには、 ANOMALY_DETECTION_BAND 関数を使用して行からチェックマークを外すか、 X アイコンを選択します。モデルを完全に削除するには、[Edit model (モデルの編集)]、[Delete model (モデルの削除)] の順に選択します。

10. (オプション) グラフ化するメトリクスを選択する際に、各メトリクスのグラフ凡例に表示する動的ラベルを指定できます。動的ラベルは、メトリクスに関する統計を表示し、ダッシュボードまたはグラフが更新されると自動的に更新されます。動的ラベルを追加するには、[グラフ化したメトリクス]、[動的ラベルを追加] の順に選択します。

    デフォルトでは、レベルに追加した動的な値はラベルの先頭に表示されます。次に、メトリクスの [ラベル] の値を選択してラベルを編集できます。詳細については、「動的ラベルを使用する」を参照してください。
11. グラフ化されたメトリクスの詳細を表示するには、凡例にマウスカーソルを合わせます。
12. 水平の注釈は、メトリクスが特定のレベルまで急上昇した時期や、メトリクスが事前定義された範囲内にあるかどうかを、グラフのユーザーがすばやく確認するのに役立ちます。水平の注釈を追加するには、[オプション] タブ、[水平の注釈の追加] の順に選択します。

    a. [ラベル] に、注釈のラベルを入力します。
    b. [値] に、水平の注釈が表示されるメトリクス値を入力します。
    c. [Fill] には、この注釈でフィルシェーディングを使用するかどうかを指定します。たとえば、対応するエリアを塗りつぶすには Above または Below を選択します。Between を指定すると、もう 1 つの Value フィールドが表示され、2 つの値の間のグラフのエリアが塗りつぶされます。
    d. [Axis] では、グラフに複数のメトリクスが含まれる場合、Value の数値が左の Y 軸または右の Y 軸のどちらに関連するメトリクスを指すかを指定します。

    注釈の塗りつぶしの色を変更するには、注釈の左側の列の色がついた四角形を選択します。

同じグラフに複数の水平注釈を追加するには、これらのステップを繰り返します。

注釈を非表示にするには、注釈の左の列のチェックボックスをオフにします。

注釈を削除するには、[アクション] の列の [x] を選択します。

13. グラフの URL を取得するには、[Actions]、[Share] を選択します。保存または共有する URL をコピーします。
14. グラフをダッシュボードに追加するには、[Actions]、[Add to dashboard] を選択します。

別のデータソースにあるメトリクスのグラフ化

CloudWatch 以外のデータソースにあるリソースをグラフ化して表示できます。これら以外のデータソースへの接続を作成する方法の詳細については、「他のデータソースにあるメトリクスへのクエリ」を参照してください。

別のデータソースにあるメトリクスをグラフ化するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで、[Metrics]、[All metrics] を選択します。
3. [マルチソースクエリ] タブを選択します。
4. [データソース] で、使用するデータソースを選択します。

   目的のデータソースへの接続をまだ作成していない場合は、[データソースの作成と管理]、[データソースの作成と管理] の順に選択します。これ以降のデータソース作成プロセスについては、「ウィザードによる事前構築済みのデータソースへの接続」を参照してください。
5. ウィザードやクエリエディタにより、クエリに必要な情報を入力するように求められます。ワークフローはデータソースごとに異なり、データソースに合わせて調整されます。例えば、Amazon Managed Service for Prometheus と Prometheus データソースの場合、PromQL クエリエディタボックスがクエリヘルパーと共に表示されます。
6. クエリの作成が完了したら、[グラフクエリ] を選択します。

   クエリからのメトリクスがグラフに入力されます。
7. (オプション) 水平の注釈は、メトリクスが特定のレベルまで急上昇した時期や、メトリクスが事前定義された範囲内にあるかどうかを、グラフのユーザーがすばやく確認するのに役立ちます。水平の注釈を追加するには、[オプション] タブ、[水平の注釈の追加] の順に選択します。

   a. [ラベル] に、注釈のラベルを入力します。
   b. [値] に、水平の注釈が表示されるメトリクス値を入力します。
   c. [Fill] には、この注釈でフィルシェーディングを使用するかどうかを指定します。たとえば、対応するエリアを塗りつぶすには `Above` または `Below` を選択します。`Between` を指

定すると、もう 1 つの Value フィールドが表示され、2 つの値の間のグラフのエリアが塗りつぶされます。

d. [Axis] では、グラフに複数のメトリクスが含まれる場合、Value の数値が左の Y 軸または右の Y 軸のどちらに関連するメトリクスを指すかを指定します。

注釈の塗りつぶしの色を変更するには、注釈の左側の列の色がついた四角形を選択します。

同じグラフに複数の水平注釈を追加するには、これらのステップを繰り返します。

注釈を非表示にするには、注釈の左の列のチェックボックスをオフにします。

注釈を削除するには、[アクション] の列の [x] を選択します。

8. (オプション) このグラフをダッシュボードに追加するには、[アクション]、[ダッシュボードに追加] の順に選択します。

グラフを更新する

グラフを更新するには

1. グラフの名前を変更するには、鉛筆アイコンを選択します。
2. 時間範囲を変更するには、事前定義済みの値を選択するか、[custom] を選択します。詳細については、「グラフの時間範囲またはタイムゾーン形式を変更する」を参照してください。
3. 統計を変更するには、[Graphed metrics] タブを選択します。列見出しまたは個々の値を選択し、統計または事前定義パーセンタイルのうち 1 つを選択するか、カスタムパーセンタイル (**p95.45** など) を指定します。
4. 期間を変更するには、[Graphed metrics] タブを選択します。列見出しまたは個々の値を選択し、次に異なる値を選択します。
5. 水平の注釈を追加するには、[グラフのオプション]、[水平の注釈の追加] の順に選択します。

a. [ラベル] に、注釈のラベルを入力します。

b. [値] に、水平の注釈が表示されるメトリクス値を入力します。

c. [Fill] には、この注釈でフィルシェーディングを使用するかどうかを指定します。たとえば、対応するエリアを塗りつぶすには Above または Below を選択します。Between を指定すると、もう 1 つの Value フィールドが表示され、2 つの値の間のグラフのエリアが塗りつぶされます。

d. [軸] で、グラフに複数のメトリクスが含まれる場合、Value の数値が左 Y 軸または右 Y 軸のどちらに関連するメトリクスを指すかを指定します。

注釈の塗りつぶしの色を変更するには、注釈の左側の列の色がついた四角形を選択します。

同じグラフに複数の水平注釈を追加するには、これらのステップを繰り返します。

注釈を非表示にするには、注釈の左の列のチェックボックスをオフにします。

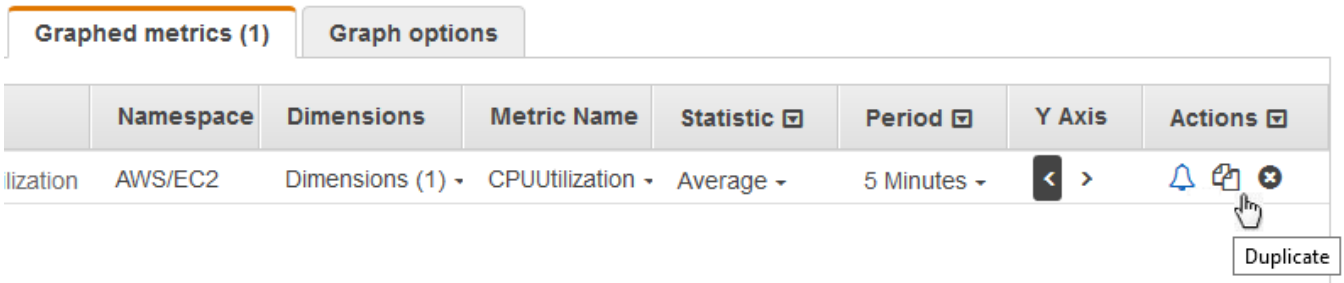
注釈を削除するには、[アクション] の列の [x] を選択します。

6. 更新間隔を変更するには、[更新オプション] を選択し、次に [自動更新] を選択するか、[1 分]、[2 分]、[5 分] または [15 分] を選択します。

メトリクスの複製

メトリクスを複製するには

1. [グラフ化したメトリクス] タブを選択します。
2. [Actions] で、[Duplicate] アイコンを選択します。



3. 必要に応じて複製メトリクスを更新します。

2 つのグラフを 1 つに結合する

2 つの異なるグラフを 1 つに結合すると、結果のグラフには両方のメトリクスが表示されます。これは、さまざまなグラフに各種メトリクスが表示され、それらを組み合わせたい場合や、さまざまな地域のメトリクスを含む単一のグラフを簡単に作成したい場合に便利です。

1 つのグラフを別のグラフに結合するには、結合するグラフの URL または JSON ソースを使用します。

2 つのグラフを 1 つに結合するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. 別のグラフに結合するグラフを開きます。このためには、[メトリクス]、[すべてのメトリクス] を選択し、グラフ化するメトリクスを選択します。または、ダッシュボードを開いて、グラフの右上にあるメニューから [メトリクスで開く] を選択して、ダッシュボード上のグラフの 1 つを開くこともできます。
3. グラフを開いたら、次のいずれかを実行します。
   - ブラウザバーから URL をコピーします。
   - [ソース] タブ、[コピー] の順に選択します。
4. 1 つ目のグラフに結合するグラフを開きます。
5. [メトリクス] ビューで 2 つ目のグラフを開いたら、[アクション]、[グラフをマージ] の順に選択します。
6. 事前にコピーした URL または JSON を入力し、[マージ] を選択します。
7. 結合したグラフが表示されます。左側の Y 軸は元のグラフ用で、右側の Y 軸は結合されたグラフ用です。

   > **Note**
   >
   > 結合したグラフが METRICS() 関数を使用する場合、結合されたグラフのメトリクスは、結合したグラフの METRICS() 計算には含まれません。

8. 結合したグラフをダッシュボードで保存するには、[アクション]、[ダッシュボードに追加] の順に選択します。

動的ラベルを使用する

グラフで動的ラベルを使用できます。動的ラベルは、選択したメトリクスのラベルに、動的に更新された値を追加します。次の表に示すように、ラベルにはさまざまな値を追加できます。

ラベルに表示される動的な値は、グラフに現在表示されている時間範囲に基づきます。このラベルの動的な部分は、ダッシュボードまたはグラフが更新されると、自動的に更新されます。

検索式で動的ラベルを使用すると、検索によって返されたすべてのメトリクスに動的ラベルが適用されます。

CloudWatch コンソールを使用して、ラベルへの動的値の追加、ラベルの編集、ラベル列内の動的値の位置の変更、その他のカスタマイズを行うことができます。

動的ラベル

動的ラベル内では、メトリクスのプロパティに関連する次の値を使用できます。

動的ラベルのライブ値	説明
${AVG}	グラフに現在表示されている時間範囲における値の平均。
${DATAPOINT_COUNT}	グラフに現在表示されている時間範囲内のデータポイントの数。
${FIRST}	グラフに現在表示されている時間範囲内の最も古いメトリクス値。
${FIRST_LAST_RANGE}	グラフに現在表示されている最も古いデータポイントと最新のデータポイントのメトリクス値の差。
${FIRST_LAST_TIME_RANGE}	グラフに現在表示されている最も古いデータポイントと最新のデータポイント間の絶対時間範囲。
${FIRST_TIME}	グラフに現在表示されている時間範囲内の最も古いデータポイントのタイムスタンプ。
${FIRST_TIME_RELATIVE}	現在と、グラフに現在表示されている時間範囲内の最も古いデータポイントのタイムスタンプとの絶対時間差。
${LABEL}	メトリクスのデフォルトラベルの表現。
${LAST}	グラフに現在表示されている時間範囲における最新のメトリクス値。
${LAST_TIME}	グラフに現在表示されている時間範囲内の最新のデータポイントのタイムスタンプ。
${LAST_TIME_RELATIVE}	現在と、グラフに現在表示されている時間範囲内の最新のデータポイントのタイムスタンプとの絶対時間差。

動的ラベルのライブ値	説明
${MAX}	グラフに現在表示されている時間範囲における最大値。
${MAX_TIME}	グラフに現在表示されているデータポイントのうち、メトリクス値が最大のデータポイントのタイムスタンプ。
${MAX_TIME_RELATIVE}	現在と、現在グラフに表示されているデータポイントのうち、最大値を持つデータポイントのタイムスタンプの絶対時間差。
${MIN}	グラフに現在表示されている時間範囲における最小値。
${MIN_MAX_RANGE}	グラフに現在表示されているデータポイントのうち、メトリクス値が最大のデータポイントと最小のデータポイント間のメトリクス値の差。
${MIN_MAX_TIME_RANGE}	グラフに現在表示されているデータポイントのうち、メトリクス値が最大のデータポイントと最小のデータポイント間の絶対時間範囲。
${MIN_TIME}	グラフに現在表示されているデータポイントのうち、メトリクス値が最小のデータポイントのタイムスタンプ。
${MIN_TIME_RELATIVE}	現在と、現在グラフに表示されているデータポイントのうち、最小値を持つデータポイントのタイムスタンプの絶対時間差。
${PROP('AccountId')}	メトリクスの AWS アカウント ID。
${PROP('AccountLabel')}	CloudWatch のクロスアカウントオブザーバビリティで、このメトリクスを所有するソースアカウントに指定されたラベル。
${PROP('Dim.*dimension_name*')}	指定されたディメンションの値。*dimension_name* はディメンション名に置き換えてください (大文字と小文字は区別されます)。
${PROP('MetricName')}	メトリクスの名前。
${PROP('Namespace')}	メトリクスの名前空間。
${PROP('Period')}	メトリクスの期間 (秒単位)。

動的ラベルのライブ値	説明
${PROP('Region')}	メトリクスが発行される AWS リージョン。
${PROP('Stat')}	グラフ化されているメトリクスの統計。
${SUM}	グラフに現在表示されている時間範囲における値の合計。

例えば、検索式 **SEARCH(' {AWS/Lambda, FunctionName} Errors ', 'Sum')** では、Lambda 関数ごとに `Errors` を確認します。ラベルを `[max: ${MAX} Errors for Function Name ${LABEL}]` として設定すると、各メトリクスのラベルは [max: ***number*** Errors for Function Name ***Name***] となります。

ラベルには、動的な値を 6 つまで追加できます。各ラベル内では `${LABEL}` プレースホルダーを 1 回のみ使用できます。

グラフの時間範囲またはタイムゾーン形式を変更する

このセクションでは、CloudWatch メトリクスのグラフ上の日付、時刻、タイムゾーン形式を変更する方法について説明します。また、グラフを拡大して特定の時間範囲を適用する方法について説明します。グラフの作成の詳細については、「メトリクスをグラフ化する」を参照してください。

> **Note**
>
> ダッシュボードの時間範囲が、ダッシュボード上のグラフに使用されている期間よりも短い場合、次が実行されます。
>
> - ダッシュボードの時間範囲よりも長い場合でも、そのウィジェットの 1 つの期間に対応するデータ量が表示されるように、グラフが変更されます。これにより、グラフ上に少なくとも 1 つのデータポイントが表示されるようになります。
> - このデータポイントの期間の開始時刻は、少なくとも 1 つのデータポイントが表示されるように逆算して調整されます。

相対時間範囲を設定する

New interface

グラフに相対時間範囲を指定するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで、[Metrics] (メトリクス)、[All metrics] (すべてのメトリクス) の順に選択します。画面の右上隅で、1 時間から 1 週間まで ([1h] (1 時間)、[3h] (3 時間)、[12h] (12 時間)、[1d] (1 日)、[3d] (3 日)、1w (1 週間)) の事前定義済みの時間範囲を 1 つ選択できます。また、[Custom] (カスタム) を選択して独自の時間範囲を設定することもできます。
3. [Custom] (カスタム) を選択してから、ボックスの左上隅にある [Relative] (相対値) タブを選択します。時間範囲は、[Minutes] (分)、[Hours] (時間)、[Days] (日)、[Weeks] (週)、[Months] (月) を指定できます。
4. 時間範囲を指定したら、[Apply] (適用) を選択します。

Original interface

グラフに相対時間範囲を指定するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで、[Metrics] (メトリクス)、[All metrics] (すべてのメトリクス) の順に選択します。画面の右上隅で、1 時間から 1 週間まで ([1h] (1 時間)、[3h] (3 時間)、[12h] (12 時間)、[1d] (1 日)、[3d] (3 日)、1w (1 週間)) の事前定義済みの時間範囲を 1 つ選択できます。また、[Custom] (カスタム) を選択して独自の時間範囲を設定することもできます。
3. [Custom] (カスタム) を選択してから、ボックスの左上隅にある [Relative] (相対値) を選択します。時間範囲は、[Minutes] (分)、[Hours] (時間)、[Days] (日)、[Weeks] (週)、[Months] (月) を指定できます。

絶対時間範囲を設定する

New interface

グラフに絶対時間範囲を指定するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。

2. ナビゲーションペインで、[Metrics] (メトリクス)、[All metrics] (すべてのメトリクス) の順に選択します。画面の右上隅で、1 時間から 1 週間まで ([1h] (1 時間)、[3h] (3 時間)、[12h] (12 時間)、[1d] (1 日)、[3d] (3 日)、1w (1 週間)) の事前定義済みの時間範囲を 1 つ選択できます。また、[Custom] (カスタム) を選択して独自の時間範囲を設定することもできます。
3. [Custom] (カスタム) を選択してから、ボックスの左上隅にある [Absolute] (絶対値) タブを選択します。カレンダーのピッカーまたはテキストフィールドボックスを使用して、時間範囲を指定します。
4. 時間範囲を指定したら、[Apply] (適用) を選択します。

Original interface

グラフに絶対時間範囲を指定するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで、[Metrics] (メトリクス)、[All metrics] (すべてのメトリクス) の順に選択します。画面の右上隅で、1 時間から 1 週間まで ([1h] (1 時間)、[3h] (3 時間)、[12h] (12 時間)、[1d] (1 日)、[3d] (3 日)、1w (1 週間)) の事前定義済みの時間範囲を 1 つ選択できます。また、[Custom] (カスタム) を選択して独自の時間範囲を設定することもできます。
3. [Custom] (カスタム) を選択してから、ボックスの左上隅にある [Absolute] (絶対値) を選択します。カレンダーのピッカーまたはテキストフィールドボックスを使用して、時間範囲を指定します。
4. 時間範囲を指定したら、[Apply] (適用) を選択します。

タイムゾーン形式を設定する

New interface

グラフのタイムゾーンを指定するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで、[Metrics] (メトリクス)、[All metrics] (すべてのメトリクス) の順に選択します。画面の右上隅で、1 時間から 1 週間まで ([1h] (1 時間)、[3h] (3 時間)、[12h] (12 時間)、[1d] (1 日)、[3d] (3 日)、1w (1 週間)) の事前定義済みの時間範囲を 1 つ選択できます。また、[Custom] (カスタム) を選択して独自の時間範囲を設定することもできます。

3. [Custom] (カスタム) を選択してから、ボックスの右上隅にあるドロップダウンを選択します。タイムゾーンは [UTC] または [Local time zone] (ローカルタイムゾーン) に変更できます。
4. 変更後、[Apply] (適用) を選択します。

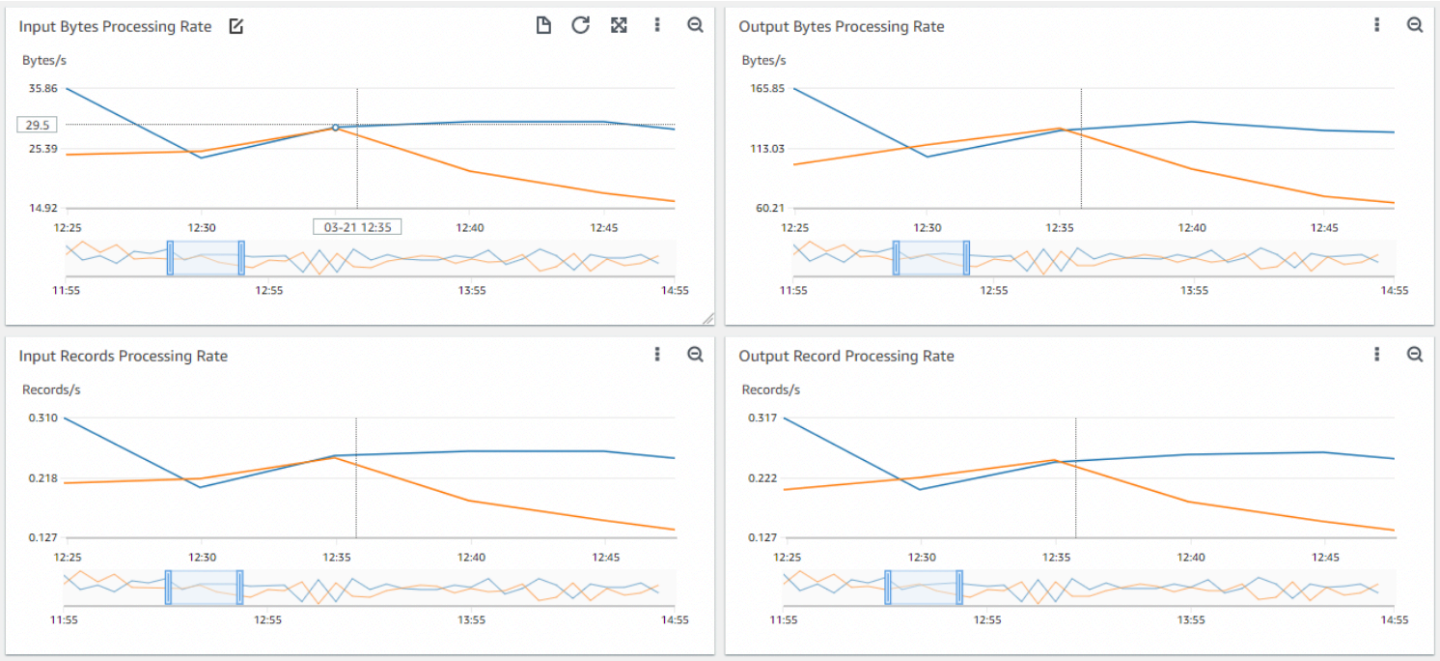
Original interface

グラフのタイムゾーンを指定するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで、[Metrics] (メトリクス)、[All metrics] (すべてのメトリクス) の順に選択します。画面の右上隅で、1 時間から 1 週間まで ([1h] (1 時間)、[3h] (3 時間)、[12h] (12 時間)、[1d] (1 日)、[3d] (3 日)、1w (1 週間)) の事前定義済みの時間範囲を 1 つ選択できます。また、[Custom] (カスタム) を選択して独自の時間範囲を設定することもできます。
3. [Custom] (カスタム) を選択してから、ボックスの右上隅にあるドロップダウンを選択します。タイムゾーンは [UTC] または [Local time zone] (ローカルタイムゾーン) に変更できます。

折れ線グラフまたは積み上げ面グラフでズームインする

CloudWatch コンソールでは、ミニマップズーム機能を使用して、ズームインビューとズームアウトビュー間を変更することなく、折れ線グラフと積み上げ面グラフのセクションに焦点を合わせることができます。例えば、ミニマップズーム機能を使用して折れ線グラフのピークに焦点を合わせ、同じタイムラインのダッシュボード内の他のメトリクスに対して、スパイクを比較できます。このセクションの手順では、ズーム機能の使用方法について説明します。



前の画像は、ズーム機能によって入力バイトの処理レートに関する折れ線グラフのスパイクに焦点を合わせながら、同じタイムラインのセクションに焦点を合わせた他の折れ線グラフをダッシュボードに表示しています。

New interface

グラフを拡大するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで、[Metrics] (メトリクス)、[All metrics] (すべてのメトリクス) の順に選択します。
3. [Browse] (参照) をクリックし、グラフ化するメトリクスを選択します。
4. [Option] (オプション) をクリックし、[Widget type] (ウィジェットタイプ) で [Line] (線) を選択します。
5. 焦点を合わせるグラフの領域を選択してドラッグし、マウスボタンを放します。
6. ズームをリセットするには、[Reset zoom] (ズームのリセット) アイコンを選択します。これは、内側にマイナス (-) 記号が付いた虫眼鏡のような見た目です。

Original interface

グラフを拡大するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで、[Metrics] (メトリクス)、[All metrics] (すべてのメトリクス) の順にクリックします。
3. [All metrics] (すべてのメトリクス) をクリックし、グラフ化するメトリクスを選択します。
4. [Graph options] (グラフのオプション) をクリックします。[Widget type] (ウィジェットタイプ) で [Line] (線) を選択します。
5. 焦点を合わせるグラフの領域を選択してドラッグし、マウスボタンを放します。
6. ズームをリセットするには、[Reset zoom] (ズームのリセット) アイコンを選択します。これは、内側にマイナス (-) 記号が付いた虫眼鏡のような見た目です。

> **Tip**
>
> 折れ線グラフまたは積み上げ面グラフを含むダッシュボードを作成済みの場合は、ダッシュボードに移動してズーム機能の使用を開始できます。

グラフの Y 軸を変更する

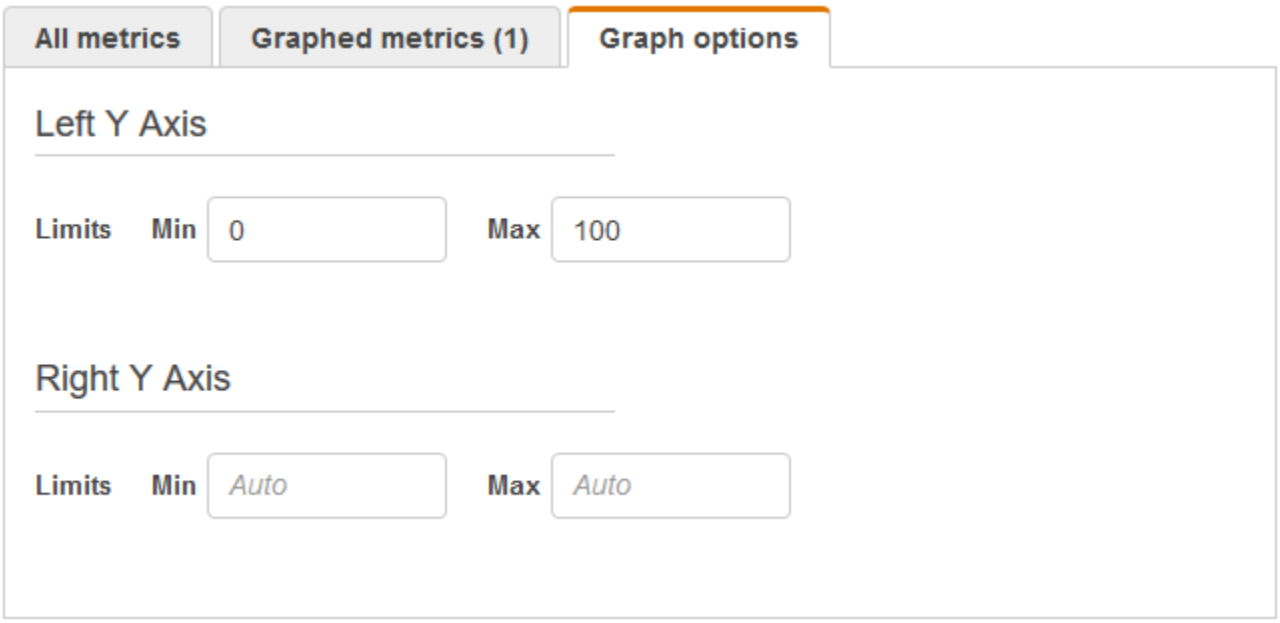
データをより良く把握するため、y 軸のカスタム境界を設定できます。たとえば、`CPUUtilization` グラフの境界を 100% に変更すると、グラフを見たときに、CPU が低い (プロットされた線がグラフの下部近くにある) か、高い (プロットされた線がグラフの上部近くにある) かが分かりやすくなります。

グラフの y 軸は、2 種類の間で切り替えることができます。これは、グラフに単位の異なるメトリクスや値の範囲が大きく異なるメトリクスが含まれているときに役立ちます。

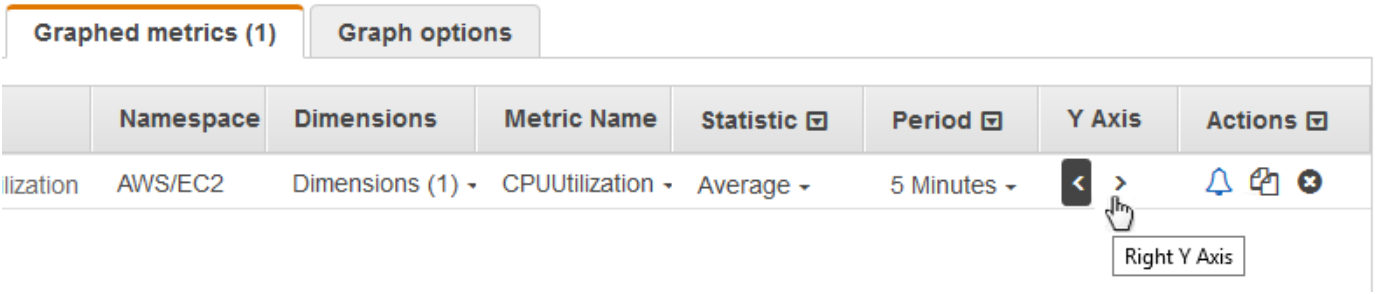
グラフの y 軸を変更するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで [Metrics (メトリクス)] を選択します。
3. メトリクス名前空間 (たとえば、[EC2]) を選択し、次にメトリクスディメンション (たとえば、[インスタンス別メトリクス]) を選択します。

4. [All metrics] タブには、その名前空間内のそのディメンションのメトリクスがすべて表示されます。メトリクスをグラフ表示するには、メトリクスの横にあるチェックボックスを選択します。
5. [Graph options] タブ上で、[Left Y Axis] の [Min] と [Max] の値を指定します。[最小] の値は、[最大] の値を上回ってはなりません。



6. 第 2 の y 軸を作成するには、[右の Y 軸] の [最小] と [最大] の値を指定します。
7. 2 つの y 軸の間を切り替えるには、[グラフ化したメトリクス] タブを選択します。[Y Axis] で、[Left Y Axis] または [Right Y Axis] を選択します。



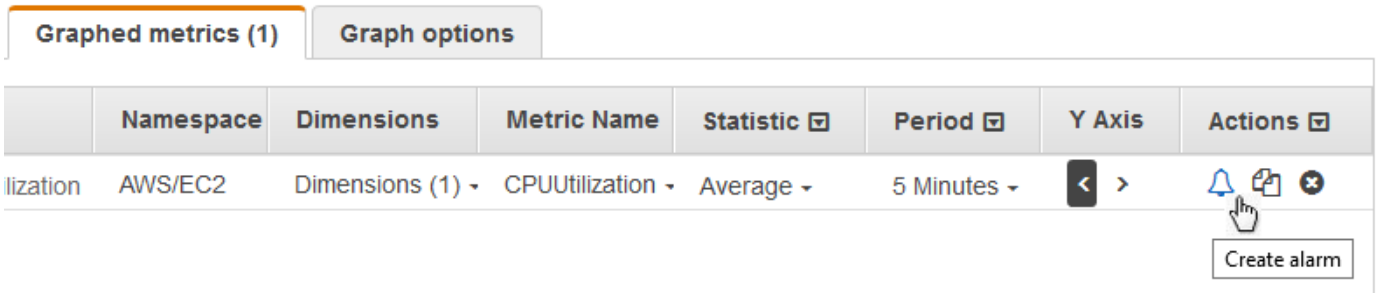
グラフのメトリクスからアラームを作成する

メトリクスをグラフ化した後、グラフのメトリクスからアラームを作成できます。こうすることで、多くのアラームフィールドの入力が自動で行われます。

グラフのメトリクスからアラームを作成するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。

2. ナビゲーションペインで [Metrics (メトリクス)] を選択します。
3. メトリクス名前空間 (たとえば、[EC2]) を選択し、次にメトリクスディメンション (たとえば、[インスタンス別メトリクス]) を選択します。
4. [All metrics] タブには、その名前空間内のそのディメンションのメトリクスがすべて表示されます。メトリクスをグラフ表示するには、メトリクスの横にあるチェックボックスを選択します。
5. メトリクスに対してアラームを作成するには、[Graphed metrics] タブを選択します。[Actions] で、アラームアイコンを選択します。



6. [Conditions (条件)] で、[Static (静的)] または [Anomaly detection (異常検出)] を選択し、静的なしきい値または異常検出モデルのどちらをアラームで使用するかを指定します。

   選択に応じて、アラーム条件の残りのデータを入力します。
7. [Additional configuration (追加設定)] を選択します。[Datapoints to alarm (アラームを発生させるデータポイント数)] で、アラームをトリガーするために ALARM 状態を維持する必要がある評価期間 (データポイント) の数を指定します。2 つの値が一致する場合は、該当する数の連続した期間でしきい値を超過したときに ALARM 状態に移行するアラームを作成します。

   N 個中 M 個のアラームを作成するには、2 番目の値よりも小さい数字を最初の値に指定します。詳細については、「アラームの評価」を参照してください。
8. [Missing data treatment (欠落データの処理)] 、一部のデータポイントが欠落しているときのアラームによる対処方法を選択します。詳細については、「CloudWatch アラームの欠落データの処理の設定」を参照してください。
9. [Next (次へ)] を選択します。
10. [通知] で、アラームが ALARM 状態、OK 状態、または INSUFFICIENT_DATA 状態のときに通知するための SNS トピックを選択します。

    同じアラーム状態または複数の異なるアラーム状態について複数の通知を送信するには、[Add notification (通知の追加)] を選択します。

    アラームの通知を送信しない場合は、 [削除] を選択します。

11. アラームに伴って Auto Scaling または EC2 アクションを実行するには、該当するボタンを選択し、アラーム状態と実行するアクションを選択します。
12. 完了したら、[次へ] を選択します。
13. アラームの名前と説明を入力します。名前には ASCII 文字のみを使用します。続いて、[次へ] を選択します。
14. [Preview and create (プレビューして作成)] で、情報と条件が正しいことを確認し、[アラームの作成] を選択します。

CloudWatch 異常検出の使用

メトリクスの異常検出を有効にすると、CloudWatch は統計アルゴリズムと機械学習アルゴリズムを適用します。これらのアルゴリズムは、システムやアプリケーションのメトリクスを継続的に分析し、正常なベースラインを決定して、ユーザーの最小限の介入で異常を検出します。

これらのアルゴリズムは、異常検出モデルを生成します。モデルは、メトリクスの正常な動作を表す想定値の範囲を生成します。

異常検出は、AWS Management Console、AWS CLI、AWS CloudFormation または AWS SDK を使用して有効にすることができます。異常検出は、AWS から提供されるメトリクスおよびカスタムメトリクスに対して有効にすることができます。CloudWatch クロスアカウントオブザーバビリティのモニタリングアカウントとして設定されたアカウントでは、モニタリングアカウントのメトリクスに加え、ソースアカウントのメトリクスに対しても異常ディテクタを作成できます。

想定値のモデルは、次の 2 つの方法で使用できます。

- メトリクスの想定値に基づいて異常検出アラームを作成します。このタイプのアラームには、アラーム状態を決定するための静的なしきい値はありません。代わりに、異常検出モデルに基づいて、メトリクスの値と想定値を比較します。

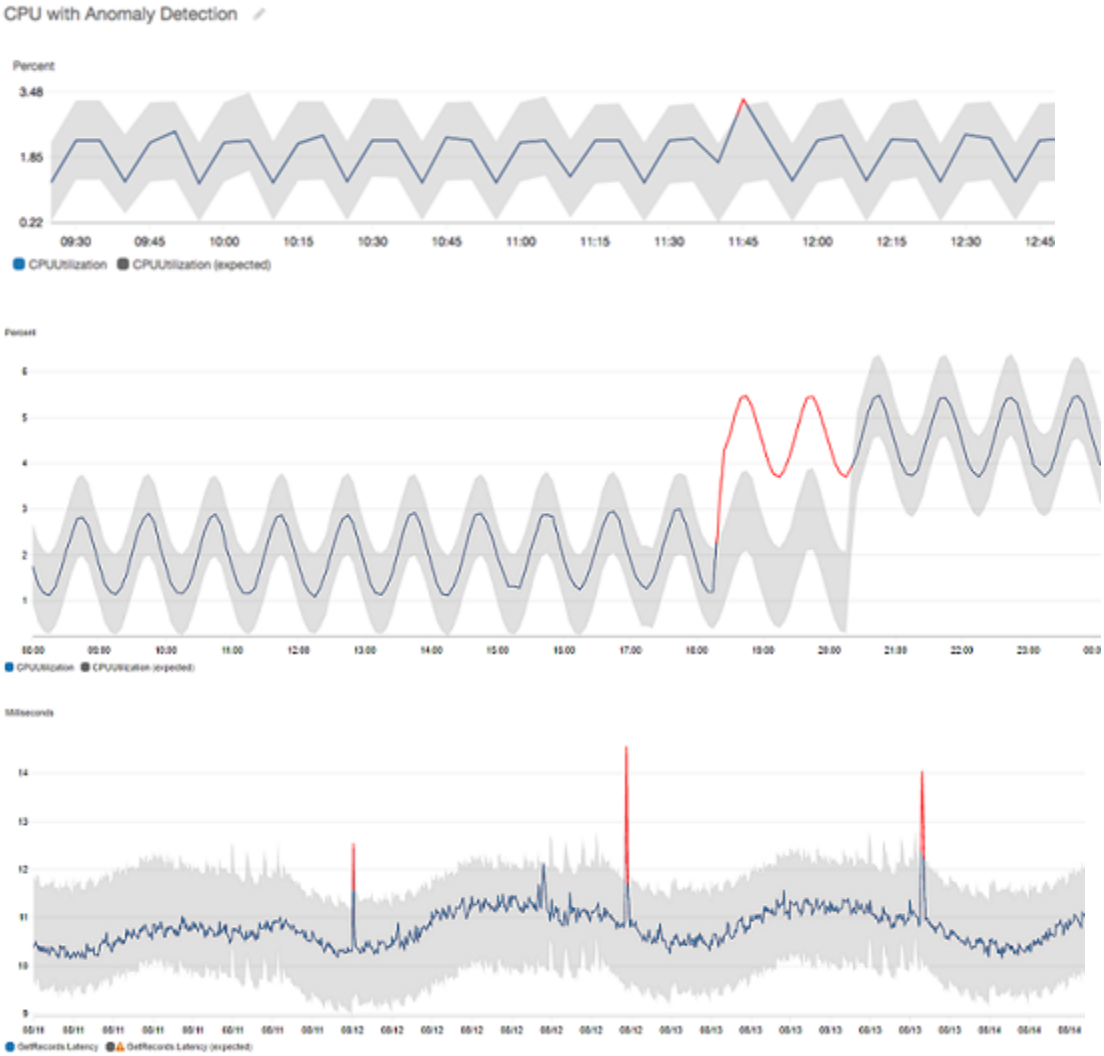
  メトリクス値が想定値の範囲を上回った場合、下回った場合、または両方の場合にアラームをトリガーすることを選択できます。

  詳細については、「異常検出に基づいて CloudWatch アラームを作成する」を参照してください。
- メトリクスデータのグラフを表示するときに、想定値をバンドとしてグラフ上に重ねます。これにより、グラフで正常な範囲から外れている値を目で確認できます。詳細については、「グラフの作成」を参照してください。

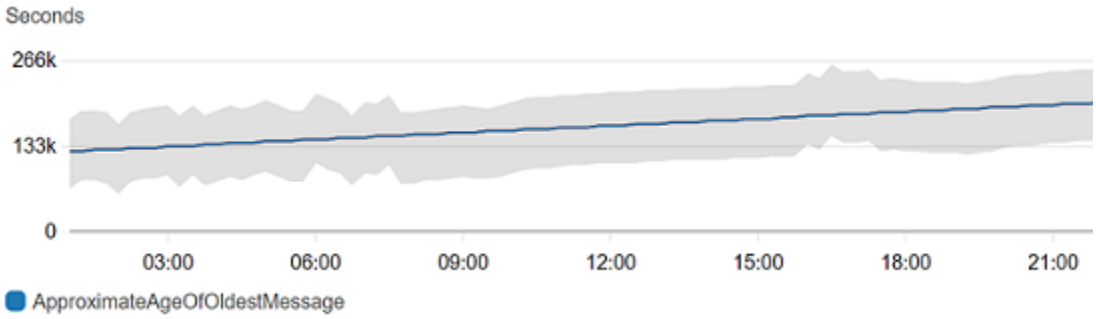
GetMetricData メトリクス数学関数で ANOMALY_DETECTION_BAND API リクエストを使用して、モデルの帯の上限と下限の値を取得することもできます。詳細については、「GetMetricData」を参照してください。

異常検出を行うグラフでは、想定値の範囲がグレーの帯で表示されます。メトリクスの実際の値がこの帯を超えると、その期間は赤で表示されます。

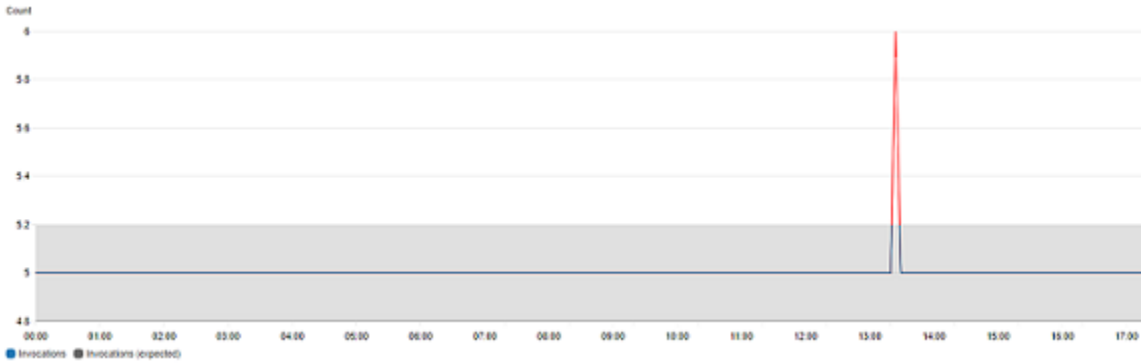
異常検出アルゴリズムは、メトリクスの季節的な変化と傾向の変化を考慮します。季節的な変化は、次の例に示すように、時間単位、日単位、週単位のいずれかになります。



より長い範囲の傾向は、下向きまたは上向きになる場合があります。



異常検出は、フラットなパターンを示すメトリクスにも適しています。



CloudWatch 異常検出の仕組み

メトリクスの異常検出を有効にすると、CloudWatch は、メトリクスの過去のデータに機械学習アルゴリズムを適用して、メトリクスの想定値のモデルを作成します。このモデルでは、メトリクスの傾向と、時間/日/週単位のパターンの両方を評価します。アルゴリズムは最大 2 週間分のメトリクスデータをトレーニングしますが、メトリクスに 2 週間分のデータが揃っていなくても、メトリクスの異常検出を有効にすることができます。

CloudWatch がモデルで使用する異常検出のしきい値を指定し、メトリクスの「正常」な値の範囲を決定します。異常検出のしきい値を高くするほど、「正常」な値の範囲が広くなります。

機械学習モデルは、メトリクスと統計に固有です。たとえば、AVG 統計を使用してメトリクスの異常検出を有効にした場合、モデルは、AVG 統計に固有になります。

CloudWatch が AWS サービスから多くの一般的なメトリクスのモデルを作成する場合、バンドが論理値の外に拡張されないようにします。例えば、EC2 インスタンスの `MemoryUtilization` のバンドは 0 から 100 の間で維持され、負にならない CloudFront `Requests` は、追跡するバンドはゼロを下回ることはありません。

モデルを作成した後、CloudWatch の異常検出はモデルを継続的に評価し、調整を行って、可能な限り正確であることを確認します。これには、メトリクス値が時間の経過とともに変化するか、突然変

化するかを調整するためのモデルの再トレーニングが含まれます。また、季節的、スパイク、スパースなメトリクスのモデルを改善するための予測変数も含まれます。

メトリクスの異常検出を有効にした後は、必要に応じて、メトリクスの特定の期間を除外してモデルのトレーニングに使用されないように指定できます。この方法により、モデルのトレーニングにデプロイまたは他の異常なイベントが使用されないように除外でき、最も正確なモデルが作成されます。

アラームに異常検出モデルを使用すると、AWS アカウントで料金が発生します。詳細については、Amazon CloudWatch 料金表をご覧ください。

Metric Math での異常検出

Metric Math での異常検出は、メトリクスの数式の出力で異常検出アラームの作成に使用できる機能です。これらの式を使用して、異常検出バンドを可視化するグラフを作成できます。この機能では、基本的な算術関数、比較演算子、論理演算子、そしてその他のほとんどの関数がサポートされています。サポートされていない関数の詳細については、Amazon CloudWatch ユーザーガイドの「Metric Math を使用する」を参照してください。

異常検出モデルの作成方法と同様に、メトリクスの数式に基づく異常検出モデルを作成できます。CloudWatch コンソールから、メトリクスの数式に異常検出を適用し、これらの式のしきい値のタイプとして異常検出を選択できます。

> **Note**
>
> Metric Math の異常検出は、最新バージョンのメトリクスユーザーインターフェイスでのみ有効化および編集できます。新しいバージョンのインターフェイスでメトリクスの数式に基づき異常ディテクターを作成すると、古いバージョンでは表示できますが、編集できません。

Metric Math と異常検出のアラームおよびモデルの作成については、次のセクションを参照してください。

- 異常検出に基づく CloudWatch アラームの作成
- メトリクスの数式に基づく CloudWatch アラームの作成

また、`PutAnomalyDetector`、`DeleteAnomalyDetector`、および `DescribeAnomalyDetectors` と CloudWatch API を使用すると、メトリクスの数式に基づく異常検出モデルを作成、削除、検出できます。これらの API アクションについては、Amazon CloudWatch API リファレンスの次のセクションを参照してください。

- PutAnomalyDetector
- DeleteAnomalyDetector
- DescribeAnomalyDetectors

異常検出アラームの価格設定については、「Amazon CloudWatch の料金」を参照してください。

Metric Math を使用する

Metric Math により複数の CloudWatch メトリクスをクエリし、数式を使用して、これらのメトリクスに基づく新しい時系列を作成できます。作成された時系列を CloudWatch コンソールで可視化でき、ダッシュボードに追加できます。AWS Lambda メトリクスを例として使用すると、`Errors` メトリクスを `Invocations` メトリクスで除算してエラー率を得ることができます。次に、作成された時系列を CloudWatch ダッシュボードのグラフに追加します。

`GetMetricData` API オペレーションを使用して、Metric Math をプログラムで実行することもできます。詳細については、「GetMetricData」を参照してください。

CloudWatch グラフに数式を追加する

CloudWatch ダッシュボードのグラフに数式を追加できます。各グラフで使用できるメトリクスおよび表現は 500 個までに制限されているため、数式はグラフのメトリクスが 499 個以下の場合のみ追加できます。これは、グラフにすべてのメトリクスが表示されていない場合でも適用されます。

グラフに数式を追加するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. グラフを作成または編集します。グラフには、少なくとも 1 つのメトリクスが必要です。
3. [グラフ化したメトリクス] を選択します。
4. [Math expression]、[Start with empty expression] の順に選択します。式に新しい行が表示されます。
5. 新しい行の [詳細] 列に、数式を入力します。[Metric Math 構文と関数] セクションの表には、式で使用できる関数が一覧表示されています。

   メトリクスまたは別の数式の結果をこの式で式の一部として使用するには、[Id] 列に表示される値を使用します (例: [m1+m2] または [e1-MIN(e1)])。

[Id] の値は変更できます。数字、文字、アンダースコアを含めることができ、小文字で始める必要があります。[Id] を意味のある名前に変更すると、グラフをより容易に理解できます (例: [m1] や [m2] を [エラー] や [リクエスト] に変更)。

> **Tip**
>
> 数式の作成時に使用できるサポートされている関数のリストを表示するには、[Math Expression (数式)] の横にある下向き矢印を選択します。

6. 数式の [ラベル] 列に、数式の計算内容を表す名前を入力します。

   式の結果が時系列の配列である場合、これらのそれぞれの時系列が、異なる色の別々の線で表示されます。グラフのすぐ下には、グラフの各行の凡例があります。複数の時系列を作成する単一の式の場合、これらの時系列の凡例のキャプションは、**#### ########** の形式になります。たとえば、ラベルが Errors であるメトリクスと、ラベルが Filled With 0: である FILL(METRICS(), 0) 式がグラフに含まれる場合、凡例の 1 つの行は Filled With 0: Errors となります。凡例で元のメトリクスラベルのみ表示されるように、**####**は空に設定します。

   1 つの式により、グラフの時系列の配列が作成される場合、これらの各時系列に使用される色を変更することはできません。

7. 必要な式を追加した後、元のメトリクスの一部を非表示にすることで、グラフを簡素化できます。メトリクスまたは式を非表示にするには、[Id] フィールドの左にあるチェックボックスをオフにします。

Metric Math 構文と関数

以下のセクションでは、Metric Math で使用できる関数について説明します。すべての関数は大文字で記述する必要があります ([AVG] など)。また、すべてのメトリクスと数式の [Id] フィールドは小文字で始める必要があります。

数式の最終結果は単一の時系列または時系列の配列である必要があります。一部の関数では、スカラー番号が生成されます。より大規模な関数の中でこれらの関数を使用することができ、最終的には時系列が生成されます。たとえば、単一の時系列の [AVG] はスカラー数を生成するため、最終的な式の結果にはなりません。ただし、このセクションを関数 m1-AVG(m1) で使用すると、個々のデータポイントと時系列の平均値の間の相違の時系列を表示できます。

データタイプの略語

一部の関数は、特定のタイプのデータのみに有効です。次のリストにある略語は、関数のテーブルで使用され、各関数でサポートされているデータタイプを表しています。

- [S] は、スカラー数 (2、-5、または 50.25 など) を表します。
- [TS] は時系列 (時間の経過に伴う単一の CloudWatch メトリクスの一連の値) です。例えば、過去 3 日間のインスタンス `i-1234567890abcdef0` の`CPUUtilization` メトリクスなどです。
- [TS[]] は時系列の配列です (複数のメトリクスの時系列など)。
- String [] は文字列の配列です。

METRICS() 関数

METRICS() 関数は、リクエストのすべてのメトリクスを返します。数式は含まれません。

より大きな式の中で METRICS() を使用することができ、最終的には単一の時系列または時系列の配列が生成されます。たとえば、SUM(METRICS()) 式は、グラフ化されたすべてのメトリクスの値の合計である時系列 (TS) を返します。METRICS()/100 は時系列の配列を返します。それぞれが、いずれかのメトリクスを 100 で除算した各データポイントを示す時系列です。

[METRICS()] 関数を文字列と使用して、[Id] フィールドにその文字列を含むグラフ化されたメトリクスのみを返すことができます。たとえば、SUM(METRICS("errors")) 式は、[Id] フィールドに「errors」のあるグラフ化されたすべてのメトリクスの値の合計である時系列を返します。また、[SUM([METRICS(“4xx”), METRICS(“5xx”)])] を使用して、複数の文字列を一致させることができます。

基本的な算術関数

サポートされている基本的な算術関数の一覧を以下の表に示しています。時系列で欠落した値は 0 として扱われます。データポイントの値により関数がゼロ除算を行おうとする場合、データポイントは削除されます。

オペレーション	引数	例
算術演算子: + - * / ^	S, S	PERIOD(m1)/60
	S, TS	5 * m1

オペレーション	引数	例
	TS, TS	m1 - m2
	S, TS[]	SUM(100/[m1, m2])
	TS, TS[]	AVG(METRICS())
		METRICS()*100
単項減算 -	S	-5*m1
	TS	-m1
	TS[]	SUM(-[m1, m2])

比較演算子と論理演算子

比較演算子と論理演算子は、時系列のペアまたは単一のスカラー値のペアと共に使用できます。比較演算子を時系列のペアと共に使用すると、各データポイントが 0 (false) または 1 (true) の時系列が返されます。スカラー値のペアで比較演算子を使用すると、単一のスカラー値 (0 または 1) が返されます。

2 つの時系列間で比較演算子が使用され、一方の時系列だけが特定のタイムスタンプの値を持つ場合、関数はもう一方の時系列の欠落している値を 0として扱います。

論理演算子を比較演算子と組み合わせて使用すると、より複雑な関数を作成できます。

次の表は、サポートされている演算子のリストです。

演算子のタイプ	サポートされている演算子
比較演算子	==
	!=
	<=
	>=
	<

演算子のタイプ	サポートされている演算子
	>
論理演算子	AND または && OR または ||

これらの演算子がどのように使用されるかを確認するため、以下の 2 つの時系列があるとします。metric1 の値は `[30, 20, 0, 0]`、metric2 の値は `[20, -, 20, -]` であり、- はそのタイムスタンプの値がないことを示します。

式	出力
(metric1 < metric2)	0, 0, 1, 0
(metric1 >= 30)	1, 0, 0, 0
(metric1 > 15 AND metric2 > 15)	1, 0, 0, 0

Metric Math のサポートされている関数

次の表は、数式で使用できる関数を表しています。すべての関数を大文字で入力します。

数式の最終結果は単一の時系列または時系列の配列である必要があります。以下のセクションの表の一部の関数では、スカラー数が生成されます。より大規模な関数の中でこれらの関数を使用することができ、最終的には時系列が生成されます。たとえば、単一の時系列の [AVG] はスカラー数を生成するため、最終的な式の結果にはなりません。しかし、それを関数 [m1-AVG(m1)] で使用して、個々のデータポイントとそのデータポイントの平均値の間の相違の時系列を表示できます。

次の表では、例の列の例は、単一の時系列または時系列の配列を生成する式です。これらの例は、スカラー値を返す関数を、単一の時系列を生成する有効な式の一部として使用する方法を示しています。

関数	引数	戻り型*	説明	例	クロスアカウントでサポートされていますか。
ABS	TS TS[]	TS TS[]	各データポイントの絶対値を返します。	ABS(m1-m2) MIN(ABS([m1, m2])) ABS(METRICS())	✓
ANOMALY_DETECTION_BAND	TS TS、S	TS[]	指定されたメトリクスの異常検出バンドを返します。バンドは 2 つの時系列で構成されます。1 つはメトリクスの「通常」の想定値の上限を表し、もう 1 つは下限を表します。この関数は 2 つの引数を受け取ることができます。1 つ目は、バンドを作成するメトリクスの ID です。2 つ目の引数は、バンドに使用する標準偏差の数です。この引数を指定しない場合、デフォルト値の 2 が使用されます。詳細については、	ANOMALY_DETECTION_BAND(m1) ANOMALY_DETECTION_BAND(m1,4)	

関数	引数	戻り型*	説明	例	クロスアカウントでサポートされていますか。
			「CloudWatch 異常検出の使用」を参照してください。		

関数	引数	戻り型*	説明	例	クロスアカウントでサポートされていますか。
AVG	TS TS[]	S TS	単一の時系列の AVG は、メトリクス内のすべてのデータポイントの平均を表すスカラーを返します。時系列の配列の AVG は単一の時系列を返します。欠落した値は 0 として処理されます。 **Note** 関数でスカラーを返すことを計画している場合、この関数は CloudWatch アラームで使用しないことをお勧めします。例えば、`AVG(m2)` と指定します。アラームが状態を変更するかどうかを	SUM([m1,m2])/AVG(m2) AVG(METRICS())	✓

関数	引数	戻り型*	説明	例	クロスアカウントでサポートされていますか。
			評価するたびに、CloudWatch は [Evaluation Periods (評価期間)] に指定されている数よりも多くのデータポイントを取得しようとします。この関数は、追加のデータがリクエストされた場合に異なる動作をします。この機能をアラーム、特に Auto Scaling アクションが設定されているアラームで使用するには、N 個のデータポイントのうち M 個を使用するようにアラームを設定		

関数	引数	戻り型*	説明	例	クロスアカウントでサポートされていますか。
			することをお勧めします (M < N)。		
CEIL	TS TS[]	TS TS[]	各メトリクスの上限を返します。上限は、各値以上の最小の整数です。	CEIL(m1) CEIL(METRICS()) SUM(CEIL(METRICS()))	✓

関数	引数	戻り型*	説明	例	クロスアカウントでサポートされていますか。
DATAPOINT_COUNT	TS TS[]	S TS	値を報告したデータポイントの数を返します。これは、スパースメトリクスの平均を計算する場合に便利です。 **Note** CloudWatch アラームではこの関数を使用しないことをお勧めします。アラームが状態を変更するかどうかを評価するたびに、CloudWatch は [Evaluation Periods (評価期間)] に指定されている数よりも多くのデータポイントを取得しようとします。	SUM(m1) / DATAPOINT_COUNT(m1) DATAPOINT_COUNT(METRICS())	✓

関数	引数	戻り型*	説明	例	クロスアカウントでサポートされていますか。
			この関数は、追加のデータがリクエストされた場合に異なる動作をします。		

関数	引数	戻り型*	説明	例	クロスアカウントでサポートされていますか。
DB_PERF_INSIGHTS	文字列、文字列、文字列 文字列、文字列、文字列[]	TS (単一の文字列が与えられた場合) TS [] (文字列の配列を指定した場合)	Amazon RDB Service や Amazon DocumentDB (MongoDB 互換性あり) などのデータベースの Performance Insights Counter メトリクスを返します。この関数は、Performance Insights API を直接クエリして取得できるのと同じ量のデータを返します。CloudWatch でこれらのメトリックスを使用して、グラフを作成し、アラームを作成できます。 **Important** この関数を使用する場合は、データベースの一意のデータベースリソース ID を指定する必	DB_PERF_INSIGHTS(‘RDS’, ‘db-ABCDEFGHIJKLMNOPQRSTUVWXY1’, ‘os.cpuUtilization.user.avg’) DB_PERF_INSIGHTS(‘DOCDB, ‘db-ABCDEFGHIJKLMNOPQRSTUVWXY1’, [‘os.cpuUtilization.idle.avg’, ‘os.cpuUtilization.user.max’])	

関数	引数	戻り型*	説明	例	クロスアカウントでサポートされていますか。
			要があります。これはデータベース ID とは異なります。Amazon RDS コンソールでデータベースリソース ID を検索するには、DB インスタンスを選択してその詳細を表示します。そして、[Configuration (設定)] タブを選択します。[設定] セクションに [リソース ID] が表示されます。 また、DB_PERF_INSIGHTS は 1 分未満の		

関数	引数	戻り型*	説明	例	クロスアカウントでサポートされていますか。
			間隔で DBLoad メトリクスを取り込みます。 この関数で取得した Performance Insights メトリクスは CloudWatch には保存されません。したがって、クロスアカウントオブザーバビリティ、異常検出、メトリクスストリーム、メトリクスエクスプローラー、Metric Insights など一部の CloudWatch 機能は、DB_PERF_INSIGHTS で取得する Performance Insights メトリクスでは動作しません。 DB_PERF_INSIGHTS 関数を使用した 1 回のリクエストで、次の数		

関数	引数	戻り型*	説明	例	クロスアカウントでサポートされていますか。
			のデータポイントを取得できます。 • 高解像度の期間 (1 秒、10 秒、30 秒) の 1,080 のデータポイント • 標準解像度の期間 (1 分、5 分、1 時間、1 日) の 1440 のデータポイント DB_PERF_INSIGHTS 関数は、以下の期間のみをサポートします • 1 秒 • 10 秒 • 30 秒 • 1 分 • 5 分 • 1 時間 • 1 日		

関数	引数	戻り型*	説明	例	クロスアカウントでサポートされていますか。
			Amazon RDS Performance Insights カウンターメトリクス詳細については、「Performance Insights カウンターメトリクス」を参照してください。 Amazon DocumentDB Performance Insights カウンターメトリクスの詳細については、「Performance Insights for counter metrics」を参照してください。 Note DB_PERF_INSIGHTS によって 1 分未満の解像度で取得される高解像度メトリクス		

関数	引数	戻り型*	説明	例	クロスアカウントでサポートされていますか。
			は、DBLoad メトリック、またはより高い解像度で拡張モニタリングを有効にしている場合はオペレーティングシステムメトリックにのみ適用されます。Amazon RDS の拡張モニタリングの詳細については、「拡張モニタリングを使用した OS メトリクスのモニタリング」を参照してください。DB_PERF_INSIGHTS 関数を使用すると、最大 3 時間の範囲で高解像度のアラームを作成		

関数	引数	戻り型*	説明	例	クロスアカウントでサポートされていますか。
			できます。CloudWatch コンソールを使用して、DB_PERF_INSIGHTS 関数で取得したメトリクスを任意の時間範囲でグラフ化できます。		
DIFF	TS TS[]	TS TS[]	時系列の各値と、その時系列の前の値との差を返します。	DIFF(m1)	✓
DIFF_TIME	TS TS[]	TS TS[]	時系列の各値のタイムスタンプと、その時系列の前の値のタイムスタンプとの差を秒単位で返します。	DIFF_TIME(METRICS())	✓

関数	引数	戻り型*	説明	例	クロスアカウントでサポートされていますか。
FILL	TS, [S | REPEA | LINEAR TS[], [TS | S | REPEA | LINEAR	TS TS[]	時系列で欠けているを埋めます。欠けている値の予備として使用する値には、いくつかのオプションがあります。 • 予備値として使用する値を指定できます。 • 予備値として使用するメトリクスを指定できます。 • REPEAT キーワードを使用して、欠けている値を、欠けている値の前にあるメトリクスの最新の実際値で埋めることができます。 • LINEAR キーワードを使用して、欠けている値を、ギャップの最初と最後の値の間	FILL(m1,10) FILL(METRICS(), 0) FILL(METRICS(), m1) FILL(m1, MIN(m1)) FILL(m1, REPEAT) FILL(METRICS(), LINEAR)	✓

関数	引数	戻り型*	説明	例	クロスアカウントでサポートされていますか。
			に線形補間を作成する値で埋めることができます。 **Note** アラームでこの関数を使用した場合、メトリクスがわずかに遅れて公開され、最新の 1 分間のデータが記録されない問題が発生することがあります。このような場合、FILL は、欠落したデータポイントをリクエストされた値に置き換えます。これにより、メトリクスの最新のデー		

関数	引数	戻り型*	説明	例	クロスアカウントでサポートされていますか。
			タポイントが常に FILL の値になり、アラームが OK 状態または ALARM 状態のいずれかで停止することがあります。N 個中 M のアラームを使用することで、この問題を回避できます。詳細については、「アラームの評価」を参照してください。		

関数	引数	戻り型*	説明	例	クロスアカウントでサポートされていますか。
FIRST LAST	TS[]	TS	時系列の配列から最初または最後の時系列を返します。これは、SORT 関数と共に使用する場合に便利です。また、ANOMALY_DETECTION_BAND 関数から高いしきい値と低いしきい値を取得するために使用することもできます。	IF(FIRST(SORT(METRICS(), AVG, DESC))>100, 1, 0) AVG でソートされた配列の上位メトリクスを確認します。その後、データポイント値が 100 より大きいかどうかに応じて、それぞれのデータポイントに対して 1 または 0 を返します。 LAST(ANOMALY_DETECTION_BAND(m1)) は、異常予測バンドの上限を返します。	✓
FLOOR	TS TS[]	TS TS[]	各メトリクスの下限を返します。下限は、各値以下の最大の整数です。	FLOOR(m1) FLOOR(METRICS())	✓

関数	引数	戻り型*	説明	例	クロスアカウントでサポートされていますか。
IF	IF expression	TS	IF を比較演算子と共に使用して、時系列からデータポイントをフィルタリングしたり、複数の照合時系列で構成される混合時系列を作成したりします。詳細については、「IF 式の使用」を参照してください。	例については、「IF 式の使用」を参照してください。	✓
INSIGHT_RULE_METRIC	INSIGHT_RULE_METRIC(ruleName, metricName)	TS	INSIGHT_RULE_METRIC を使用して、Contributor Insights のルールから統計を抽出します。詳細については、「ルールによって生成されたメトリクスのグラフ化」を参照してください。		

関数	引数	戻り型*	説明	例	クロスアカウントでサポートされていますか。
LAMBDA	LAMBDmbdaFuionNam[, optionalarg]*)	TS TS{}	Lambda 関数を呼び出すと、CloudWatch 以外のデータソースにあるメトリクスをクエリできます。詳細については、「Lambda 関数に引数を渡す方法」を参照してください。		
LOG	TS TS[]	TS TS[]	時系列の LOG は、時系列の各値の自然対数値を返します。	LOG(METRICS())	✓
LOG10	TS TS[]	TS TS[]	時系列の LOG10 は、時系列の各値の 10 を底とする対数値を返します。	LOG10(m1)	✓

関数	引数	戻り型*	説明	例	クロスアカウントでサポートされていますか。
MAX	TS TS[]	S TS	単一の時系列の MAX は、メトリクス内のすべてのデータポイントの最大値を表すスカラーを返します。 時系列の配列を入力すると、MAX 関数は、入力として使用された時系列の中で、各データポイントの最大値で構成される時系列を作成して返します。 **Note** 関数でスカラーを返すことを計画している場合、この関数は CloudWatch アラームで使用しないことをお勧めします。例えば、MAX(m2) ア	MAX(m1)/m1 MAX(METRICS())	✓

関数	引数	戻り型*	説明	例	クロスアカウントでサポートされていますか。
			ラームが状態を変更するかどうかを評価するたびに、CloudWatch は [評価期間] に指定されている数よりも多くのデータポイントを取得しようとします。この関数は、追加のデータがリクエストされた場合に異なる動作をします。		
METRIC_COUNT	TS[]	S	時系列の配列でメトリクスの数を返します。	m1/METRIC_COUNT(METRICS())	✓

関数	引数	戻り型*	説明	例	クロスアカウントでサポートされていますか。
METRICS	null 文字列	TS[]	METRICS() 関数は、リクエストのすべての CloudWatch メトリクスを返します。数式は含まれません。 より大きな式の中で METRICS() を使用することができ、最終的には単一の時系列または時系列の配列が生成されます。 [METRICS()] 関数を文字列と使用して、[Id] フィールドにその文字列を含むグラフ化されたメトリクスのみを返すことができます。たとえば、SUM(METRICS("errors")) 式は、[Id] フィールドに「errors」のあるグラフ化されたすべてのメトリクスの値の合計である時系列	AVG(METRICS()) SUM(METRICS("errors"))	✓

関数	引数	戻り型*	説明	例	クロスアカウントでサポートされていますか。
			を返します。また、[SUM([METRICS(“4xx”), METRICS(“5xx”)])] を使用して、複数の文字列を一致させることができます。		

関数	引数	戻り型*	説明	例	クロスアカウントでサポートされていますか。
MIN	TS TS[]	S TS	単一の時系列の MIN は、メトリクス内のすべてのデータポイントの最小値を表すスカラーを返します。 時系列の配列を入力すると、MIN 関数は、入力として使用された時系列の中で、各データポイントの最小値で構成される時系列を作成して返します。 時系列の配列を入力すると、MIN 関数は、入力として使用された時系列の中で、各データポイントの最大値で構成される時系列を作成して返します。	m1-MIN(m1) MIN(METRICS())	✓

関数	引数	戻り型*	説明	例	クロスアカウントでサポートされていますか。
			Note 関数でスカラーを返すことを計画している場合、この関数は CloudWatch アラームで使用しないことをお勧めします。例えば、`MIN(m2)` アラームが状態を変更するかどうかを評価するたびに、CloudWatch は [評価期間] に指定されている数よりも多くのデータポイントを取得しようとします。この関数は、追加のデータがリクエストされた場合に異		

関数	引数	戻り型*	説明	例	クロスアカウントでサポートされていますか。
			なる動作をします。		

関数	引数	戻り型*	説明	例	クロスアカウントでサポートされていますか。

MINUTE
 	
HOUR
 	
DAY
 	
DATE
 	
MONTH
 	
YEAR
 	
EPOCH
	TS	TS	これらの関数は、時系列の期間と範囲を取り、各値がそのタイムスタンプに基づいている新しい非スパース時系列を返します。
			
			• MINUTE は、元の時系列の各タイムスタンプで UTC 分を表す、非スパース時系列 (0～59 の整数) を返します。
			• HOUR は、元の時系列の各タイムスタンプで UTC 時間を表す、非スパース時系列 (0～23 の整数) を返します。
			• DAY は、元の時系列の各タイムスタンプで UTC 曜日を表す、非スパース時系列 (1～7の整数) を返しま
				MINUTE(m1)
				
				IF(DAY(m1)<6,m1) は、月曜日から金曜日の平日のメトリクスのみを返します。
				
				IF(MONTH(m1) == 4,m1) は、4 月に公開されたメトリクスのみを返します。
					✓

関数	引数	戻り型*	説明	例	クロスアカウントでサポートされていますか。
			す。1 は月曜日を、7 は日曜日を表します。 • DATE は、元の時系列の各タイムスタンプで UTC 日付を表す非スパース時系列 (1～31の整数) を返します。 • MONTH は、元の時系列の各タイムスタンプの UTC 月を表す、非スパース時系列 (1～12の整数) を返します。1 は 1 月を表し、12 は 12 月を表します。 • YEAR は、元の時系列の各タイムスタンプで UTC 年を表す、非スパース時系列を返します。 • EPOCH は、元の時系列の各タイムスタ		

関数	引数	戻り型*	説明	例	クロスアカウントでサポートされていますか。
			ンプで UTC 時間を秒単位で表す、整数の非スパース時系列を返します。Epoch は 1970 年 1 月 1 日です。		
PERIOD	TS	S	メトリクスの期間を秒単位で返します。有効な入力は、他の式の結果ではなくメトリクスです。	m1/PERIOD(m1)	✓

関数	引数	戻り型*	説明	例	クロスアカウントでサポートされていますか。
RATE	TS TS[]	TS TS[]	メトリクスの変更のレートを秒単位で返します。これは、最新のデータポイントの値と、前のデータポイントの値の差を、2 つ値の間の時間 (秒単位) で除算して計算されます。 **Important** すパーツデータのメトリクスに RATE 関数を使用する式にアラームを設定すると、アラームの評価時に取得されるデータポイントの範囲は、データポイントが最後に公開された日時によって異なる可能性があるた	RATE(m1) RATE(METRICS())	✓

関数	引数	戻り型*	説明	例	クロスアカウントでサポートされていますか。
			め、予期しない動作が発生することがあります。		

関数	引数	戻り型*	説明	例	クロスアカウントでサポートされていますか。
REMOVE_EMPTY	TS[]	TS[]	時系列の配列から、データポイントを持たない時系列を削除します。この結果は、各時系列に少なくとも 1 つのデータポイントが含まれている時系列の配列となります。 **Note** CloudWatch アラームではこの関数を使用しないことをお勧めします。アラームが状態を変更するかどうかを評価するたびに、CloudWatch は [Evaluation Periods (評価期間)] に指定されている数よりも多くのデータポ	REMOVE_EMPTY(METRICS())	✓

関数	引数	戻り型*	説明	例	クロスアカウントでサポートされていますか。
			イントを取得しようとします。この関数は、追加のデータがリクエストされた場合に異なる動作をします。		

関数	引数	戻り型*	説明	例	クロスアカウントでサポートされていますか。
RUNNING_SUM	TS TS[]	TS TS[]	元の時系列の値から現在の合計を持つ時系列を返します。 **Note** CloudWatch アラームではこの関数を使用しないことをお勧めします。アラームが状態を変更するかどうかを評価するたびに、CloudWatch は [Evaluation Periods (評価期間)] に指定されている数よりも多くのデータポイントを取得しようとします。この関数は、追加のデータがリクエストされた	RUNNING_SUM([m1,m2])	✓

関数	引数	戻り型*	説明	例	クロスアカウントでサポートされていますか。
			場合に異なる動作をします。		

関数	引数	戻り型*	説明	例	クロスアカウントでサポートされていますか。
SEARCH	Search 式	1 つ以上の TS	指定した検索条件に一致する 1 つ以上の時系列を返します。[SEARCH] 関数を使用することで、1 つの式で複数の関連する時系列をグラフに追加できます。グラフは、後に追加され、検索基準に一致する新しいメトリクスが含まれるように、動的に更新されます。詳細については、「グラフで検索式を使用する」を参照してください。 検索式に基づくアラームを作成することはできません。これは、検索式が複数の時系列を返し、数式に基づくアラームは 1 つの時系列しか監視できないためです。		✓

関数	引数	戻り型*	説明	例	クロスアカウントでサポートされていますか。
			CloudWatch クロスアカウントオブザーバビリティでモニターリングアカウントにサインインしている場合、SEARCH 関数はソースアカウントとモニターリングアカウントのメトリクスを検索します。		
SERVICE_QUOTA	使用状況メトリクスである TS	TS	特定の使用状況メトリクスのサービスクォータを返します。これを使用して、現在の使用状況とクォータの比較を可視化し、クォータに近づいたときに警告するアラームを設定できます。詳細については、「AWS 使用状況メトリクス」を参照してください。		✓

関数	引数	戻り型*	説明	例	クロスアカウントでサポートされていますか。
SLICE	(TS[], S, S) または (TS[], S)	TS[] TS	時系列の配列の一部を取得します。これは、SORT と組み合わせた場合に特に便利です。たとえば、時系列の配列から最上位の結果を除外できます。 2 つのスカラー引数を使用して、返される時系列のセットを定義できます。2 つのスカラーは、返す配列の先頭 (その値を含む) と末尾 (その値を含まない) を定義します。配列はゼロから始まるため、配列の最初の時系列は時系列 0 です。または、値を 1 つだけ指定すると、その値で始まるすべての時系列が CloudWatch により返されます。	SLICE(SORT(METRICS(), SUM, DESC), 0, 10) は、SUM 値が最も大きいリクエスト内のメトリクスの配列から 10 個のメトリクスを返します。 SLICE(SORT(METRICS(), AVG, ASC), 5) は、AVG 統計でメトリクスの配列をソートし、AVG が最も低い 5 以外のすべての時系列を返します。	✓

関数	引数	戻り型*	説明	例	クロスアカウントでサポートされていますか。
SORT	(TS[], FUNCT SORT_(R) (TS[], FUNCT SORT_(R, S)	TS[]	指定した関数に従って時系列の配列をソートします。使用する関数は、AVG、MIN、MAX、S のいずれかです。ソート順は、昇順の場合は ASC (最小値が先頭)、または大きい値を最初にソートする場合は DESC です。必要に応じて、制限として機能するソート順の後に数値を指定できます。たとえば、制限を 5 に指定すると、ソートから上位 5 個の時系列のみが返されます。 この数学関数をグラフに表示すると、グラフ内の各メトリクスのラベルもソートされ、番号が付けられます。	SORT(METRICS(), AVG, DESC, 10) は、各時系列の平均値を計算して、ソートの開始時に最大値を持つ時系列をソートし、平均値が最も高い 10 個の時系列のみを返します。 SORT(METRICS(), MAX, ASC) は、メトリクスの配列を MAX 統計でソートし、そのすべてを昇順で返します。	✓

関数	引数	戻り型*	説明	例	クロスアカウントでサポートされていますか。
STDDEV	TS TS[]	S TS	単一の時系列の STDDEV は、メトリクス内のすべてのデータポイントの標準偏差を表すスカラーを返します。時系列の配列の STDDEV は単一の時系列を返します。 **Note** 関数でスカラーを返すことを計画している場合、この関数は CloudWatch アラームで使用しないことをお勧めします。例えば、STDDEV(m2) アラームが状態を変更するかどうかを評価するたびに、CloudWatch は [評	m1/STDDEV(m1) STDDEV(METRICS())	✓

関数	引数	戻り型*	説明	例	クロスアカウントでサポートされていますか。
			価期間] に指定されている数よりも多くのデータポイントを取得しようとします。この関数は、追加のデータがリクエストされた場合に異なる動作をします。		

関数	引数	戻り型*	説明	例	クロスアカウントでサポートされていますか。
SUM	TS TS[]	S TS	単一の時系列の SUM は、メトリクス内のすべてのデータポイントの合計値を表すスカラーを返します。時系列の配列の SUM は単一の時系列を返します。 **Note** 関数でスカラーを返すことを計画している場合、この関数は CloudWatch アラームで使用しないことをお勧めします。例えば、`SUM(m1)` と指定します。アラームが状態を変更するかどうかを評価するたびに、CloudWatch	SUM(METRICS())/SUM(m1) SUM([m1,m2]) SUM(METRICS("errors"))/SUM(METRICS("requests"))*100	✓

関数	引数	戻り型*	説明	例	クロスアカウントでサポートされていますか。
			は [Evaluation Periods (評価期間)] に指定されている数よりも多くのデータポイントを取得しようとします。この関数は、追加のデータがリクエストされた場合に異なる動作をします。		
TIME_SERIES	S	TS	すべての値がスカラー引数に設定されている非スパース時系列を返します。	TIME_SERIES(MAX(m1)) TIME_SERIES(5*AVG(m1)) TIME_SERIES(10)	✓

*スカラー数を返す関数の使用は無効です。式の最終結果はすべて単一の時系列または時系列の配列である必要があります。代わりに、これらの関数を、時系列を返すより大きな式の一部として使用します。

IF 式の使用

IF を比較演算子と共に使用して、時系列からデータポイントをフィルタリングしたり、複数の照合時系列で構成される混合時系列を作成したりします。

IF は、次の引数を使用します。

```
IF(condition, trueValue, falseValue)
```

条件は、条件の値が正か負かにかかわらず、条件データポイントの値が 0 の場合は FALSE、条件の値がその他の値の場合は TRUE と評価されます。条件が時系列の場合、タイムスタンプごとに個別に評価されます。

有効な構文を次に示します。これらの構文ごとに、出力は単一の時系列です。

- IF(TS ***Comparison Operator*** S, S | TS, S | TS)

  > **Note**
  >
  > TS comparison operator S が TRUE で、metric2 に対応するデータポイントがない場合、出力は 0 になります。

- IF(TS, TS, TS)
- IF(TS, S, TS)
- IF(TS, TS, S)
- IF(TS, S, S)
- IF(S, TS, TS)

次のセクションでは、これらの構文の詳細と例を示します。

IF(TS ***Comparison Operator*** S, scalar2 | metric2, scalar3 | metric3)

対応する出力時系列値:

- TS ***Comparison Operator*** S が TRUE の場合、これには値として scalar2 または metric2 が含まれます。
- TS ***Comparison Operator*** S が FALSE の場合、これには値として scalar3 または metric3 が含まれます。

- は、TS **#####**が TRUE で、metric2 の対応するデータポイントが存在しない場合、値 0 を持ちます。
- は、TS **#####**が FALSE で、metric3 の対応するデータポイントが存在しない場合、値 0 を持ちます。
- 空の時系列です (対応するデータポイントが metric3 に存在しない場合、または scalar3/metric3 が式から省略されている場合)。

IF(metric1, metric2, metric3)

metric1 の各データポイントの対応する出力時系列の値は次のとおりです。

- 値は metric2 です (対応するデータポート metric1 が TRUE の場合)。
- 値は metric3 です (対応するデータポート metric1 が FALSE の場合)。
- 値は 0 です (対応するデータポイント metric1 が TRUE で、対応するデータポイントが metric2 に存在しない場合)。
- は、metric1 の対応するデータポイントが FALSE で、対応するデータポイントが metric3 に存在しない場合、削除されます
- は、metric1 の対応するデータポイントが FALSE で、対応するデータポイントが metric3 に存在しない場合、削除されます。
- は、metric1 の対応するデータポイントが存在しない場合、削除されます。

次の表に、この構文の例を示します。

メトリクスまたは関数	値
(metric1)	[1, 1, 0, 0, -]
(metric2)	[30, -, 0, 0, 30]
(metric3)	[0, 0, 20, -, 20]
IF(metric1, metric2, metric3)	[30, 0, 20, 0, -]

IF(metric1, scalar2, metric3)

metric1 の各データポイントの対応する出力時系列の値は次のとおりです。

- 値は scalar2 です (対応するデータポート metric1 が TRUE の場合)。
- 値は metric3 です (対応するデータポート metric1 が FALSE の場合)。
- 削除されます (対応するデータポイント metric1 が FALSE で、対応するデータポイントが metric3 に存在しない場合、または metric3 が式から省略されている場合)。

メトリクスまたは関数	値
(metric1)	[1, 1, 0, 0, -]
scalar2	5
(metric3)	[0, 0, 20, -, 20]
IF(metric1, scalar2, metric3)	[5, 5, 20, -, -]

IF(metric1, metric2, scalar3)

metric1 の各データポイントの対応する出力時系列の値は次のとおりです。

- 値は metric2 です (対応するデータポート metric1 が TRUE の場合)。
- 値は scalar3 です (対応するデータポート metric1 が FALSE の場合)。
- 値は 0 です (対応するデータポイント metric1 が TRUE で、対応するデータポイントが metric2 に存在しない場合)。
- metric1 の対応するデータポイントが存在しない場合は削除されます。

メトリクスまたは関数	値
(metric1)	[1, 1, 0, 0, -]
(metric2)	[30, -, 0, 0, 30]
scalar3	5
IF(metric1, metric2, scalar3)	[30, 0, 5, 5, -]

IF(scalar1, metric2, metric3)

対応する出力時系列値:

- 値は metric2 です (scalar1 が TRUE の場合)。
- 値は metric3 です (scalar1 が FALSE の場合)。
- 空の時系列です (metric3 が式から省略されている場合)。

IF 式のユースケース例

次の例は、IF 関数の考えれられる使用方法を示しています。

- メトリクスの低い値のみを表示するには

  IF(metric1<400, metric1)
- メトリクスの各データポイントを 2 つの値のいずれかに変更して、元のメトリクスの相対的に大きい値と小さい値を表示するには

  IF(metric1<400, 10, 2)
- レイテンシーがしきい値を超える各タイムスタンプに 1 を表示し、その他のすべてのデータポイントに 0 を表示するには

  IF(latency>threshold, 1, 0)

GetMetricData API オペレーションでメトリクス数式を使用する

`GetMetricData` を使用して数式を使用した計算を実行でき、1 回の API コールでメトリクスデータの大規模バッチを取得できます。詳細については、「GetMetricData」を参照してください。

Metric Math での異常検出

Metric Math の異常検出は、単一のメトリクスおよびメトリクスの数式からの出力により、異常検出アラームを作成する際に使用できる機能です。これらの式を使用して、異常検出バンドを可視化するグラフを作成できます。この機能では、基本的な算術関数、比較演算子、論理演算子、そしてその他のほとんどの関数がサポートされています。

Metric Math の異常検出では、以下の関数はサポートされていません。

- 同じ行に複数の ANOMALY_DETECTION_BAND を含む表現。

- 10 より多くのメトリクスまたは数式を含む表現。
- METRICS 式を含む表現。
- SEARCH 関数を含む表現。
- DP_PERF_INSIGHTS 関数を使用する表現。
- 異なる期間のメトリクスを使用する表現。
- Metric Math で高分解能のメトリクスを入力として使用する異常検出機能。

この機能の詳細については、Amazon CloudWatch ユーザーガイドの「CloudWatch 異常検出の使用」を参照してください。

グラフで検索式を使用する

検索式は、CloudWatch グラフに追加できる数式の一種です。検索式を使用すると、グラフに複数の関連メトリクスをすばやく追加することができます。また、グラフを最初に作成したときにそれらのメトリクスが存在しない場合でも、それらのディスプレイに適切なメトリクスを自動的に追加する動的グラフを作成することもできます。

たとえば、リージョン内のすべてのインスタンスの `AWS/EC2 CPUUtilization` メトリクスを表示する検索式を作成することができます。新しいインスタンスを後に起動する場合、その新しいインスタンスの `CPUUtilization` はグラフに自動的に追加されます。

検索式をグラフで使用すると、メトリクス名、名前空間、ディメンション名、ディメンション値で検索式が検索されます。複雑性が高く、強力な検索には、ブール演算子を使用できます。検索式で検索できるのは、過去 2 週間以内にデータを報告したメトリクスだけです。

検索式に基づくアラームを作成することはできません。これは、検索式が複数の時系列を返し、数式に基づくアラームは 1 つの時系列しか監視できないためです。

CloudWatch クロスアカウントオブザーバビリティでモニターリングアカウントを使用している場合、検索式はそのモニターリングアカウントにリンクされているソースアカウントのメトリクスを検出できます。

トピック

- CloudWatch 検索式の構文
- CloudWatch 検索式の例
- 検索式で CloudWatch グラフを作成する

CloudWatch 検索式の構文

有効な検索式の形式は次のとおりです。

```
SEARCH(' {Namespace, DimensionName1, DimensionName2, ...} SearchTerm', 'Statistic')
```

次に例を示します。

```
SEARCH('{AWS/EC2,InstanceId} MetricName="CPUUtilization"', 'Average')
```

- 検索語 SEARCH の後のクエリの最初の部分 (中括弧で囲まれている) は、検索されるメトリクススキーマです。メトリックスキーマには、メトリクス名前空間と 1 つ以上のディメンション名が含まれています。検索クエリにメトリクスキーマを含めることは、オプションです。指定した場合、メトリクスキーマには名前空間が含まれている必要があり、その名前空間で有効な 1 つ以上のディメンション名をオプションで含めることができます。

  名前空間またはディメンション名にスペースまたは英数字以外の文字が含まれている場合を除き、メトリクスキーマ内で引用符を使用する必要はありません。その場合は、それらの文字を含む名前を二重引用符で囲む必要があります。
- また、SearchTerm もオプションですが、有効な検索には、メトリクスキーマか SearchTerm、またはその両方を含める必要があります。SearchTerm には通常、アカウント ID、メトリクス名、またはディメンション値が 1 つまたは複数含まれます。SearchTerm には、部分一致および完全一致で検索する用語を複数含めることができます。ブール演算子を含めることもできます。

  SearchTerm でアカウント ID を使用しても、CloudWatch クロスアカウントオブザーバビリティのモニターリングアカウントとして設定されているアカウントでのみ有効です。SearchTerm のアカウント ID の構文は `:aws.AccountId = "444455556666"` です。`'LOCAL'` を使用して `:aws.AccountId = 'LOCAL'` のようにモニターリングアカウント自体を指定することもできます。

  詳細については、「CloudWatch のクロスアカウントオブザーバビリティ」を参照してください。

  SearchTerm には、1 つ以上の指定子 (この例の MetricName= など) を含めることができますが、指定子の使用は必須ではありません。

  メトリクススキーマと SearchTerm は、単一引用符のペアで一緒に囲む必要があります。
- Statistic は、有効な CloudWatch 統計の名前です。一重引用符で囲む必要があります。詳細については、「統計」を参照してください。

前の例では、ディメンション名として AWS/EC2 を持つすべてのメトリクスについて InstanceId 名前空間を検索します。検出したすべての CPUUtilization メトリクスと、Average 統計を示すグラフを返します。

検索式で検索できるのは、過去 2 週間以内にデータを報告したメトリクスだけです。

検索式の制限

検索式の最大クエリサイズは 1,024 文字です。1 つのグラフに最大 100 つの検索式を含めることができます。グラフには最大 500 の時系列を表示できます。

CloudWatch 検索式: トークン分割

SearchTerm を指定すると、検索機能でトークンが検索されます。トークンは、CloudWatch で、完全なメトリクス名、ディメンション名、ディメンション値、名前空間から自動的に生成される部分文字列です。CloudWatch により、元の文字列のキャメルケース文字で区別されたトークンが生成されます。数字は新しいトークンの開始部分として、英数字以外の文字は区切り文字として機能します。その結果、英数字以外の文字の前後にトークンが作成されます。

同じタイプのトークンの区切り文字の連続文字列は 1 つのトークンになります。

トークンはすべて小文字で生成されます。生成されたトークンの例を次の表に示します。

元の文字列	生成されたトークン
CustomCount1	customcount1 , custom, count, 1
SDBFailure	sdbfailure , sdb, failure
Project2-trial333	project2trial333 , project, 2, trial, 333

CloudWatch 検索式: 部分一致

SearchTerm を指定すると、検索語もトークン分割されます。CloudWatch は、部分一致に基づいてメトリクスを検索します。部分一致は、検索語から生成された単一のトークンと、メトリクス名、名前空間、ディメンション名、またはディメンション値から生成された単一のトークンとの一致を表します。

単一のトークンに一致する部分一致検索では、大文字と小文字は区別されません。たとえば、次のいずれかの検索語を使用して CustomCount1 メトリクスを返すことができます。

- count
- Count
- COUNT

ただし、検索語 couNT は、cou と NT にトークン分割されるため、検索語に couNT を使用しても、CustomCount1 は見つかりません。

元の名前に連続して現れる複数のトークンである複合トークンにも一致するように検索することもできます。複合トークンと一致させるには、大文字と小文字を区別して検索します。たとえば、元の語が CustomCount1 で、CustomCount や Count1 と検索すると一致しますが、customcount や count1 では一致しません。

CloudWatch 検索式: 完全一致

完全一致を必要とする検索語の部分を二重引用符で囲むことで、検索語の完全一致のみを検索するように検索を定義することができます。これらの二重引用符は、検索語全体で使用される単一引用符で囲まれています。たとえば、CustomCount1 が MyNamespace という名前の名前空間にメトリクス名、ディメンション名、ディメンション値として存在する場合は、**SEARCH(' {MyNamespace}, "CustomCount1" ', 'Maximum')** で正確な文字列が検索されます。ただし、**SEARCH(' {MyNamespace}, "customcount1" ', 'Maximum')** または **SEARCH(' {MyNamespace}, "Custom" ', 'Maximum')** と検索した場合、この文字列は見つかりません。

部分一致の語と完全一致の語を単一の検索式に組み合わせることができます。例えば、**SEARCH(' {AWS/NetworkELB, LoadBalancer} "ConsumedLCUs" OR flow ', 'Maximum')** では、ConsumedLCUs という名前の Elastic Load Balancing メトリクスと、トークン flow を含むすべての Elastic Load Balancing メトリクスまたはディメンションが返されます。

次の例のように、英数字以外の文字やスペースなどの特殊文字を含む名前を見つけるには、完全一致を使用することもお勧めします。

```
SEARCH(' {"My Namespace", "Dimension@Name"}, "Custom:Name[Special_Characters" ',
 'Maximum')
```

CloudWatch 検索式: メトリクススキーマの除外

これまでに示した例にはすべて、中括弧で囲まれたメトリクスキーマが含まれています。スキーマメトリクスを省略する検索も有効です。

たとえば、**SEARCH(' "CPUUtilization" ', 'Average')** では、文字列 CPUUtilization の完全一致であるメトリクス名、ディメンション名、ディメンション値、名前空間がすべて返ります。AWS のメトリクス名前空間では、Amazon EC2、Amazon ECS、SageMaker などの複数のサービスのメトリクスを含めることができます。

この検索を 1 つの AWS サービスに絞り込むには、ベストプラクティスとして、メトリクススキーマの名前空間と必要なディメンションを指定することをお勧めします。以下に例を示します。この場合、検索は AWS/EC2 名前空間に絞り込まれているため、そのようなメトリクスのディメンション値として CPUUtilization を定義した場合は、他のメトリクスの結果が返ります。

```
SEARCH(' {AWS/EC2, InstanceType} "CPUUtilization" ', 'Average')
```

または、以下の例のように、SearchTerm に名前空間を追加することもできます。ただし、この例では、カスタムのディメンション名または値であっても、AWS/EC2 に一致します。

```
SEARCH(' "AWS/EC2" MetricName="CPUUtilization" ', 'Average')
```

CloudWatch 検索式: 検索でプロパティ名を指定する

以下の "CustomCount1" の完全一致検索では、その名称と正確に一致するすべてのメトリクスが返ります。

```
SEARCH(' "CustomCount1" ', 'Maximum')
```

ただし、CustomCount1 のディメンション名、ディメンション値、または名前空間のメトリクスも返ります。検索をさらに構造化するには、検索で見つけるオブジェクトのタイプのプロパティ名を指定します。次の例では、すべての名前空間を検索し、CustomCount1 という名前のメトリクスが返ります。

```
SEARCH(' MetricName="CustomCount1" ', 'Maximum')
```

また、名前空間とディメンション名/値をプロパティ名として使用することもできます。以下に例を示します。これらの例のうち、1 つ目は、部分一致検索でもプロパティ名を使用できることを示しています。

```
SEARCH(' InstanceType=micro ', 'Average')
```

```
SEARCH(' InstanceType="t2.micro" Namespace="AWS/EC2" ', 'Average')
```

CloudWatch 検索式: 英数字以外の文字

英数字以外の文字は区切り文字として機能し、メトリクスの名前、ディメンション、名前空間、および検索語をトークンに分割する場所を示します。語がトークン分割されると、英数字以外の文字は取り除かれ、トークンには表示されません。たとえば、Network-Errors_2 では、トークン network、errors、および 2 が生成されます。

検索語には、英数字以外の文字を含めることができます。これらの文字が検索語に含まれている場合は、部分一致で複合トークンを指定できます。たとえば、次の検索ではすべて、Network-Errors-2 または NetworkErrors2 という名前のメトリクスが見つかります。

```
network/errors
network+errors
network-errors
Network_Errors
```

完全な値で検索を行う場合、完全一致検索で使用される英数字以外の文字は、検索対象の文字列に含まれる正しい文字である必要があります。たとえば、Network-Errors-2 を検索する場合、"Network-Errors-2" と検索すると一致しますが、"Network_Errors_2" では一致しません。

完全一致検索を行う場合、以下の文字はバックスラッシュを使って エスケープする必要があります。

```
" \ ( )
```

たとえば、完全一致でメトリクス名 Europe\France Traffic(Network) を検索するには、検索語に **"Europe\\France Traffic\(Network\)"** を使用します。

CloudWatch 検索式: ブール演算子

ブール演算子 AND、OR、NOT を SearchTerm 内で使用して検索することができます。ブール演算子は、検索語全体を囲むために使用する単一引用符で囲まれています。ブール演算子では、大文字と小文字は区別されるため、and、or、および not はブール演算子として有効ではありません。

AND を検索で明示的に使用することができます。たとえば、**SEARCH('{AWS/EC2,InstanceId} network AND packets', 'Average')** のように使用します。AND 演算子がない場合、検索語間にブール値を使用しないと明示的に検索されないため、**SEARCH(' {AWS/EC2,InstanceId} network packets ', 'Average')** では、同じ検索結果が得られます。

検索結果からデータのサブセットを除外するには NOT を使用します。たとえば、**SEARCH(' {AWS/EC2,InstanceId} MetricName="CPUUtilization" NOT i-1234567890123456 ', 'Average')** では、すべてのインスタンスの CPUUtilization が返ります (インスタンス i-1234567890123456 は除く)。また、唯一の検索語として、NOT を使用することもできます。たとえば、**SEARCH('NOT Namespace=AWS ', 'Maximum')** では、すべてのカスタムメトリクス (AWS を含まない名前空間を含むメトリクス) が得られます。

クエリでは、複数の NOT 句を使用できます。たとえば、**SEARCH(' {AWS/EC2,InstanceId} MetricName="CPUUtilization" NOT "ProjectA" NOT "ProjectB" ', 'Average')** では、リージョン内のすべてのインスタンスの CPUUtilization が返ります (ただし、ディメンション値 ProjectA または ProjectB が含まれるものは除く)。

強力で詳細な検索を行うには、ブール演算子を組み合わせることができます。以下に例を示します。演算子をグループ化するには括弧を使用します。

次の 2 つの例ではいずれも、EC2 と EBS 名前空間から、ReadOps を含むメトリクス名がすべて返ります。

```
SEARCH(' (EC2 OR EBS) AND MetricName=ReadOps ', 'Maximum')
```

```
SEARCH(' (EC2 OR EBS) MetricName=ReadOps ', 'Maximum')
```

次の例では、前の検索は、ProjectA を含む結果のみに絞り込まれます。その結果、ディメンションの値になります。

```
SEARCH(' (EC2 OR EBS) AND ReadOps AND ProjectA ', 'Maximum')
```

次の例では、ネストされたグループ化を使用します。すべての関数からの Errors の Lambda メトリクスと、ProjectA または ProjectB を含む名前の関数の Invocations を返します。

```
SEARCH(' {AWS/Lambda,FunctionName} MetricName="Errors" OR (MetricName="Invocations" AND
 (ProjectA OR ProjectB)) ', 'Average')
```

CloudWatch 検索式: 数式の使用

検索式は、グラフ内の数式で使用できます。

例えば、**SUM(SEARCH(' {AWS/Lambda, FunctionName} MetricName="Errors" ', 'Sum'))** は、すべての Lambda 関数の Errors メトリクスの合計を返します。

検索式と数式に個別の行を使用すると、より有用な結果が得られる場合があります。たとえば、グラフで次の 2 つの式を使用するとします。最初の行には、各 Lambda 関数の各 Errors 行が表示されます。この数式の ID は、e1 です。2 行目には、すべての関数からのエラーの合計を示す別の行が追加されています。

```
SEARCH(' {AWS/Lambda, FunctionName}, MetricName="Errors" ', 'Sum')
SUM(e1)
```

CloudWatch 検索式の例

次の例は、検索式の使用方法と構文を示しています。リージョン内のすべてのインスタンスで CPUUtilization を検索してから、バリエーションを見てみましょう。

この例では、リージョン内の各インスタンスの 1 行が表示されており、CPUUtilization 名前空間の AWS/EC2 メトリクスを示します。

```
SEARCH(' {AWS/EC2,InstanceId} MetricName="CPUUtilization" ', 'Average')
```

InstanceId を InstanceType に変更すると、リージョンで使用された各インスタンスタイプの 1 行を示すようにグラフが変更されます。各タイプのすべてのインスタンスのデータは、そのインスタンスタイプの 1 行に集約されます。

```
SEARCH(' {AWS/EC2,InstanceType} MetricName="CPUUtilization" ', 'Average')
```

次の例では、CPUUtilization をインスタンスタイプ別に集計しており、文字列 micro を含む各インスタンスタイプの 1 行が表示されます。

```
SEARCH('{AWS/EC2,InstanceType} InstanceType=micro MetricName="CPUUtilization" ',
 'Average')
```

この例は、前の例を絞り込んでおり、InstanceType は、t2.micro instance の完全一致検索に変更されます。

```
SEARCH('{AWS/EC2,InstanceType} InstanceType="t2.micro" MetricName="CPUUtilization" ',
 'Average')
```

次の検索では、クエリの {metric schema} 部分が削除されるため、すべての名前空間の CPUUtilization メトリクスがグラフに表示されます。これにより、グラフには、さまざまなディ

メンションに沿って計算された、AWS の各サービスの CPUUtilization メトリクスの複数の行が含まれるため、多くの結果が返ることがあります。

```
SEARCH('MetricName="CPUUtilization" ', 'Average')
```

これらの結果を絞り込むために、2 つの特定のメトリクス名前空間を指定できます。

```
SEARCH('MetricName="CPUUtilization" AND ("AWS/ECS" OR "AWS/ES") ', 'Average')
```

各クエリに指定できるメトリクスキーマは 1 つのみであるため、前の例は、1 つの検索クエリで特定の複数の名前空間の検索を実行する唯一の方法です。ただし、構造をさらに追加するには、次の例のようにグラフで 2 つのクエリを使用できます。また、この例では、Amazon ECS のデータを集計するために使用するディメンションを指定して、構造をさらに追加しています。

```
SEARCH('{AWS/ECS ClusterName}, MetricName="CPUUtilization" ', 'Average')
SEARCH(' {AWS/EBS} MetricName="CPUUtilization" ', 'Average')
```

次の例では、ConsumedLCUs という名前の Elastic Load Balancing メトリクスと、トークン `flow` を含むすべての Elastic Load Balancing メトリクスまたはディメンションが返されます。

```
SEARCH('{AWS/NetworkELB, LoadBalancer} "ConsumedLCUs" OR flow ', 'Maximum')
```

次の例では、ネストされたグループ化を使用します。すべての関数からの `Errors` の Lambda メトリクスと、`ProjectA` または `ProjectB` を含む名前の関数の `Invocations` を返します。

```
SEARCH('{AWS/Lambda,FunctionName} MetricName="Errors" OR (MetricName="Invocations" AND
 (ProjectA OR ProjectB)) ', 'Average')
```

以下の例では、すべてのカスタムメトリクスが表示されていますが、AWS のサービスで生成されたメトリクスは除きます。

```
SEARCH('NOT Namespace=AWS ', 'Average')
```

次の例では、メトリクス名、名前空間、ディメンション名、および名前の一部として文字列 `Errors` を含むディメンション値を含むメトリクスが表示されています。

```
SEARCH('Errors', 'Average')
```

次の例では、検索対象を完全一致に絞り込みます。たとえば、この検索では、メトリクス名 Errors は一致しますが、ConnectionErrors や errors という名前のメトリクスには一致しません。

```
SEARCH(' "Errors" ', 'Average')
```

次の例は、検索語のメトリクススキーマ部分にスペースまたは特殊文字を含む名前を指定する方法を示しています。

```
SEARCH('{"Custom-Namespace", "Dimension Name With Spaces"}, ErrorCount ', 'Maximum')
```

CloudWatch クロスアカウントオブザーバビリティ検索式の例

CloudWatch クロスアカウントオブザーバビリティの例

CloudWatch クロスアカウントオブザーバビリティでモニターリングアカウントとして設定されているアカウントにサインインしている場合は、SEARCH 関数を使用して指定したソースアカウントのメトリクスを返すことができます。詳細については、「CloudWatch のクロスアカウントオブザーバビリティ」を参照してください。

次の例では、アカウント ID 111122223333 のアカウントの Lambda メトリクスをすべて取得します。

```
SEARCH(' AWS/Lambda :aws.AccountId = "111122223333" ', 'Average')
```

次の例では、111122223333 と 777788889999 の 2 つのアカウントから AWS/EC2 メトリクスをすべて取得します。

```
SEARCH(' AWS/EC2 :aws.AccountId = ("111122223333" OR "777788889999") ', 'Average')
```

次の例では、ソースアカウント 111122223333 とモニタリングアカウント自体から AWS/EC2 メトリクスをすべて取得します。

```
SEARCH(' AWS/EC2 :aws.AccountId = ("111122223333" OR 'LOCAL') ', 'Average')
```

次の例では、InstanceId ディメンションを持つアカウント 444455556666 から MetaDataToken メトリクスの SUM を取得します。

```
SEARCH('{AWS/EC2,InstanceId} :aws.AccountId=444455556666 MetricName=\"MetadataNoToken
\"','Sum')
```

検索式で CloudWatch グラフを作成する

CloudWatch コンソールで、グラフをダッシュボードに追加する際に検索機能にアクセスするか、[メトリクス] ビューを使用することができます。

検索式に基づくアラームを作成することはできません。これは、検索式が複数の時系列を返し、数式に基づくアラームは 1 つの時系列しか監視できないためです。

検索式をしようしてグラフを既存のダッシュボードに追加するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで [ダッシュボード] を選択し、ダッシュボードを選択します。
3. [ウィジェットの追加] を選択します。
4. [ライン] または [スタックされたエリア] を選択し、[設定] を選択します。
5. [グラフ化したメトリクス] タブで、[Add a math expression (数式の追加)] を検索します。
6. [詳細] に、必要な検索式を入力します。たとえば、**`SEARCH('{AWS/EC2,InstanceId} MetricName="CPUUtilization"', 'Average')`** と指定します。
7. (オプション) 別の検索式または数式をグラフに追加するには、[Add a math expression (数式の追加)] を選択します。
8. (オプション) 検索式を追加した後で、各メトリクスのグラフ凡例に示す動的ラベルを指定できます。動的ラベルは、メトリクスに関する統計を表示し、ダッシュボードまたはグラフが更新されると自動更新されます。動的ラベルを追加するには、[グラフ化したメトリクス]、[Dynamic labels (動的ラベル)] の順に選択します。

   デフォルトでは、レベルに追加した動的な値はラベルの先頭に表示されます。次に、メトリクスの [ラベル] の値をクリックしてラベルを編集できます。詳細については、「動的ラベルを使用する」を参照してください。
9. (オプション) 単一のメトリクスをグラフに追加するには、[すべてのメトリクス] タブを選択し、必要なメトリクスにドリルダウンします。
10. (オプション) グラフに表示される時間範囲を変更するには、グラフ上部の [カスタム] または [カスタム] の左にある期間のいずれかを選択します。
11. (オプション) 水平の注釈を使用すると、メトリクスが特定のレベルまで急上昇した時点や、メトリクスが事前定義した範囲内にあるかどうかをすばやく確認できます。水平の注釈を追加するには、[グラフのオプション]、[水平の注釈の追加] の順に選択します。

    a. [ラベル] に、注釈のラベルを入力します。

b. [値] に、水平の注釈が表示されるメトリクス値を入力します。

c. [Fill] には、この注釈でフィルシェーディングを使用するかどうかを指定します。たとえば、対応するエリアを塗りつぶすには Above または Below を選択します。Between を指定すると、もう 1 つの Value フィールドが表示され、2 つの値の間のグラフのエリアが塗りつぶされます。

d. [軸] では、グラフに複数のメトリクスが含まれる場合、Value の数値が左 Y 軸または右 Y 軸のどちらに関連するメトリクスを指すかを指定します。

注釈の塗りつぶしの色を変更するには、注釈の左側の列の色がついた四角形を選択します。

同じグラフに複数の水平注釈を追加するには、これらのステップを繰り返します。

注釈を非表示にするには、注釈の左の列のチェックボックスをオフにします。

注釈を削除するには、[アクション] の列の [x] を選択します。

12. (オプション) 垂直の注釈は、運用イベントや、デプロイの開始/終了などのマイルストーンをグラフにマークするために役立ちます。垂直の注釈を追加するには、[グラフのオプション]、[垂直の注釈の追加] の順に選択します。

a. [ラベル] に、注釈のラベルを入力します。注釈で日時のみを表示するには、[ラベル] フィールドを空白のままにします。

b. [日付] に、垂直の注釈が表示される日時を指定します。

c. [範囲] で、フィルシェーディングを垂直の注釈の前で使用するか、後で使用するか、または 2 つの注釈の間に使用するかを指定します。たとえば、対応するエリアを塗りつぶすには Before または After を選択します。Between を指定すると、もう 1 つの Date フィールドが表示され、2 つの値の間のグラフのエリアが塗りつぶされます。

同じグラフに複数の垂直注釈を追加するには、これらのステップを繰り返します。

注釈を非表示にするには、注釈の左の列のチェックボックスをオフにします。

注釈を削除するには、[アクション] の列の [x] を選択します。

13. [ウィジェットの作成] を選択します。

14. [ダッシュボードの保存] を選択します。

[メトリクス] ビューを使用して、検索されたメトリクスを使用するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで [Metrics (メトリクス)] を選択します。
3. [検索] フィールドに、検索するトークン (例: **cpuutilization t2.small**) を入力します。

   検索条件に一致する結果が表示されます。
4. 検索条件に一致するすべてのメトリクスをグラフ化するには、[グラフの検索] を選択します。

   または

   検索を絞り込むには、検索結果に表示された名前空間のいずれかを選択します。
5. 検索を絞り込むための名前空間を選択した場合は、以下のように行うことができます。

   a. 1 つ以上のメトリクスをグラフ化するには、各メトリクスの横にあるチェックボックスを選択します。すべてのメトリクスを選択するには、テーブルの見出し行にあるチェックボックスを選択します。

   b. 検索を絞り込むには、メトリクス名にカーソルを合わせ、[検索に追加] または [これのみを検索] を選択します。

   c. メトリクスのヘルプを表示するには、メトリクス名を選択し、[What is this?] (これは何ですか？) を選択します。

   選択したメトリクスがグラフに表示されます。
6. (オプション) 検索バーのボタンのいずれかを選択して、検索語のその部分を編集します。
7. (オプション) グラフをダッシュボードに追加するには、[アクション]、[ダッシュボードに追加] の順に選択します。

メトリクスの統計を取得する

CloudWatch 統計定義

統計とは、メトリクスデータを指定した期間で集計したものです。メトリクスの統計をグラフ化または取得するときに、各統計値の計算に使用するPeriod (5 分など) を指定します。例えば、Periodが 5 分である場合、合計は 5 分間に収集されたすべてのサンプル値の合計です。一方、Minimumは、5 分期間中に収集された最小値です。

CloudWatch では、メトリクスの次の統計を使用できます。

- SampleCount は、期間中のデータポイント数です。
- Sum は、その期間中に収集されたすべてのデータポイントの値の合計です。
- Average は、指定した期間の Sum/SampleCount の値です。
- Minimum は、指定された期間に認められた最小値です。
- Maximum は、指定された期間に認められた最大値です。
- Percentile (p) は、データセットにおける値の相対的な位置を示します。例えば、p95 は 95 パーセンタイルであり、期間の 95 パーセントのデータがこの値を下回っており、5 パーセントのデータがこの値を上回っていることを意味します。パーセンタイルにより、メトリクスデータの分布をよく理解することができます。
- Trimmed mean (TM) は、指定された 2 つの境界の間にあるすべての値の平均です。平均の計算時には、境界の外側の値は無視されます。境界は、0 ~ 100 の 1 つまたは 2 つの数値 (小数点以下 10 桁まで) で定義します。数値には、絶対値またはパーセンテージを指定できます。例えば、tm90 は、最も高い値を持つデータポイントの 10% を削除した後の平均を計算します。TM(2%:98%) は、最低 2% のデータポイントと最高 2% のデータポイントを削除した後の平均を計算します。TM(150:1000) は、150 以下、または 1000 以上のすべてのデータポイントを削除した後の平均を計算します。
- Interquartile mean (IQM) は四分位範囲のトリム平均、または値の中央の 50% です。これは、TM(25%:75%) に相当します。
- Winsorized mean (WM) は、トリム平均に似ています。ただし、ウィンソライズされた平均値では、境界の外側にある値は無視されず、代わりに適切な境界のエッジにある値と等しいと見なされます。この正規化の後、平均が計算されます。境界は、0 ~ 100 の 1 つまたは 2 つの数値 (小数点以下 10 桁まで) で定義します。例えば、wm98 は、最高値の 2% を 98パーセンタイルの値と等しくなるように処理しながら、平均を計算します。WM(10%:90%) は、データポイントの最高 10% を 90% 境界の値として扱い、データポイントの最低 10% を 10% 境界の値として扱います。
- Percentile rank (PR) は、固定しきい値を満たす値の割合です。例えば、PR(:300) は、300 以下の値を持つデータポイントの割合を返します。PR(100:2000) は、100 ~ 2000 の間の値を持つデータポイントの割合を返します。

  パーセンタイルランクは下限には含まれず、上限には含まれます。
- Trimmed count (TC) は、トリミング平均統計の選択した範囲内のデータポイント数です。例えば、tc90 は、値の最高 10% に含まれるデータポイントを含まないデータポイントの数を返します。TC(0.005:0.030) は、0.005 (排他的) から 0.030 (包括的) の間の値を持つデータポイントの数を返します。

- Trimmed sum (TS) は、トリミング平均統計の選択した範囲内のデータポイントの値の合計です。これは、(トリム平均値) * (トリム数) に相当します。例えば、ts90 は、値の最高 10% に含まれるデータポイントを含まないデータポイントの合計を返します。TS(80%:) は、データポイント値の合計を返します。値の範囲の最下位 80% の値を持つデータポイントは含まれません。

> **Note**
>
> Trimmed Mean、Trimmed Count、Trimmed Sum、および Winsorized Mean では、2 つの境界をパーセンテージではなく固定値として定義すると、計算には上側の境界に等しい値は含まれますが、下側の境界に等しい値は含まれません。

構文

Trimmed Mean、Trimmed Count、Trimmed Sum、および Winsorized Mean には、次の構文規則が適用されます。

- 1 つまたは 2 つの数値にパーセント記号を付けた括弧を使用すると、指定した 2 つの百分位数の間にあるデータセット内の値として使用する境界が定義されます。例えば、TM(10%:90%) では、10 番目から 90 番目の百分位数の間の値のみが使用されます。TM(:95%) では、設定されたデータの最下端から 95 パーセンタイルまでの値が使用され、最も高い値を持つデータポイントの 5% は無視されます。
- パーセント記号を付けずに 1 つまたは 2 つの数値に括弧を使用すると、指定した明示的な値の間に収まるデータセット内の値として使用する境界が定義されます。例えば、TC(80:500) は、80 (排他的) から 500 (包括的) の値のみを使用します。TC(:0.5) では、0.5 以下の値のみが使用されます。
- 括弧を付けずに 1 つの数値を使用すると、指定した百分位数よりも大きいデータポイントは無視され、パーセンテージを使用して計算されます。例えば、tm99 は、最も高い値を持つデータポイントの 1% を無視して平均を計算します。これは、TM(:99%) と同じです。
- Trimmed mean、Trimmed Count、Trimmed Sum、および Winsorized Mean は、範囲を指定するときに、すべて大文字を使用して省略できます。例えば、TM(5%:95%)、TM (100:200)、またはTM(:95%) などです。例えば、tm99 など、数字を 1 つだけ指定する場合は、小文字のみを使用して省略できます。

統計ユースケース

- Trimmed mean は、ウェブページのレイテンシーなど、大きなサンプルサイズのメトリクスで最も便利です。例えば、tm99 は、ネットワークの問題や人為的なエラーから生じる可能性のある極端に高い外れ値を無視し、一般的なリクエストの平均レイテンシーに対してより正確な数を与えます。同様に、TM(10%:) は、キャッシュヒットによるレイテンシー値など、レイテンシー値の最低 10% を無視します。そして、TM(10%:99%) は、こうしたタイプの外れ値の両方を除外します。レイテンシーのモニタリングには、トリミング平均を使用することをお勧めします。
- トリミング平均を使用するときは常に、トリミングされたカウントを監視し、トリミングされた平均計算で使用される値の数が統計的に有意になるほど十分であることを確認することをお勧めします。
- 百分位ランクを使用すると、値を範囲の「ビン」に入れることができ、これを使用して手動でヒストグラムを作成できます。これを行うには、値を PR(:1)、PR(1:5)、PR(5:10)、および PR(10:) などさまざまなビンに分割します。これらのビンのそれぞれを棒グラフとして可視化すると、ヒストグラムが得られます。

  パーセンタイルランクは下限には含まれず、上限には含まれます。

百分位数対トリミング平均

p99 などの百分位数および tm99 などのトリミングされた平均値は、類似しているが、同一ではない値を測定します。p99 および tm99 の両方は、外れ値と見なされる最も高い値を持つデータポイントの 1% を無視します。その後、p99 は残りの 99% の最大値となり、tm99 は残りの 99% の平均値となります。ウェブリクエストのレイテンシーを見ている場合は、p99 は、外れ値を無視して、最悪のカスタマーエクスペリエンスを示しますが、tm99 は、外れ値を無視して、平均的なカスタマーエクスペリエンスを示します。

Trimed mean は、カスタマーエクスペリエンスの最適化を検討しているかどうかを確認するのに適したレイテンシー統計です。

百分位数、調整平均値、およびその他の統計量を使用するための要件

CloudWatch は、以下の統計を計算するために、raw データポイントを必要とします。

- パーセンタイル
- トリム平均
- 四分位平均

- ウィンソライズ平均
- トリム合計
- トリム数
- パーセンタイルランク

raw データの代わりに統計セットを使用してカスタム統計のデータを発行する場合は、以下の条件のいずれかが真である場合のみ、このデータのこれらのタイプの統計を取得できます。

- 統計セットの SampleCount 値は 1 で、Min、Max、Sum はすべて等しくなります。
- Min と Max は等しく、Sum は Min に SampleCount を乗算した値に等しくなります。

次の AWS サービスには、パーセンタイルこれらのタイプの統計をサポートするメトリクスが含まれています。

- API Gateway
- Application Load Balancer
- Amazon EC2
- Elastic Load Balancing
- Kinesis
- Amazon RDS

さらに、これらのタイプの統計は、メトリクス値が負の数値のメトリクスに対して使用することはできません。

次の例では、EC2 インスタンスなどのリソースに対する CloudWatch メトリクスの統計を取得する方法を説明します。

例

- 特定のリソースの統計を取得する
- 統計をリソース間で集計する
- Auto Scaling グループ別に統計を集計する
- Amazon マシンイメージ (AMI) で統計を集計する

特定のリソースの統計を取得する

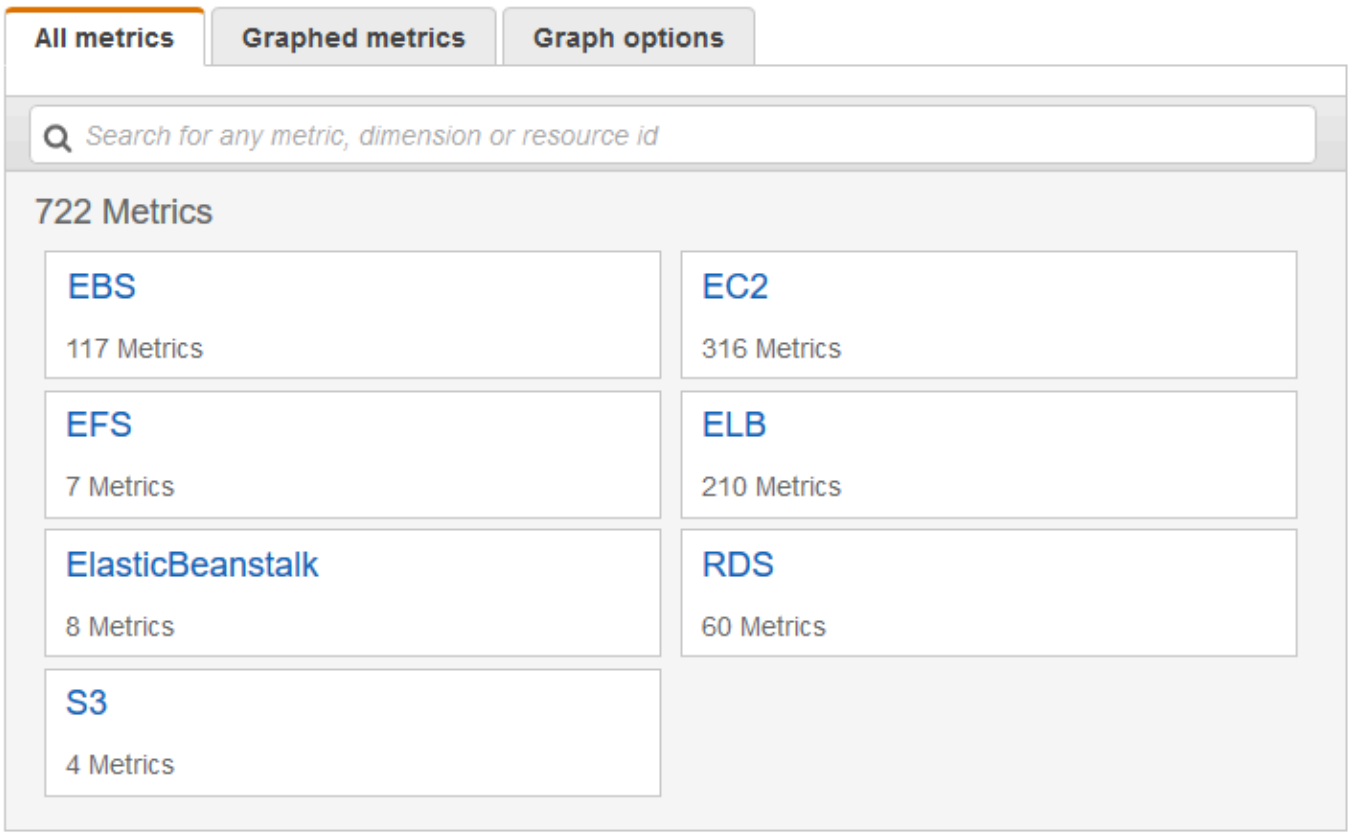
次の例では、特定の EC2 インスタンスの最大 CPU 使用率を特定する方法を説明します。

要件

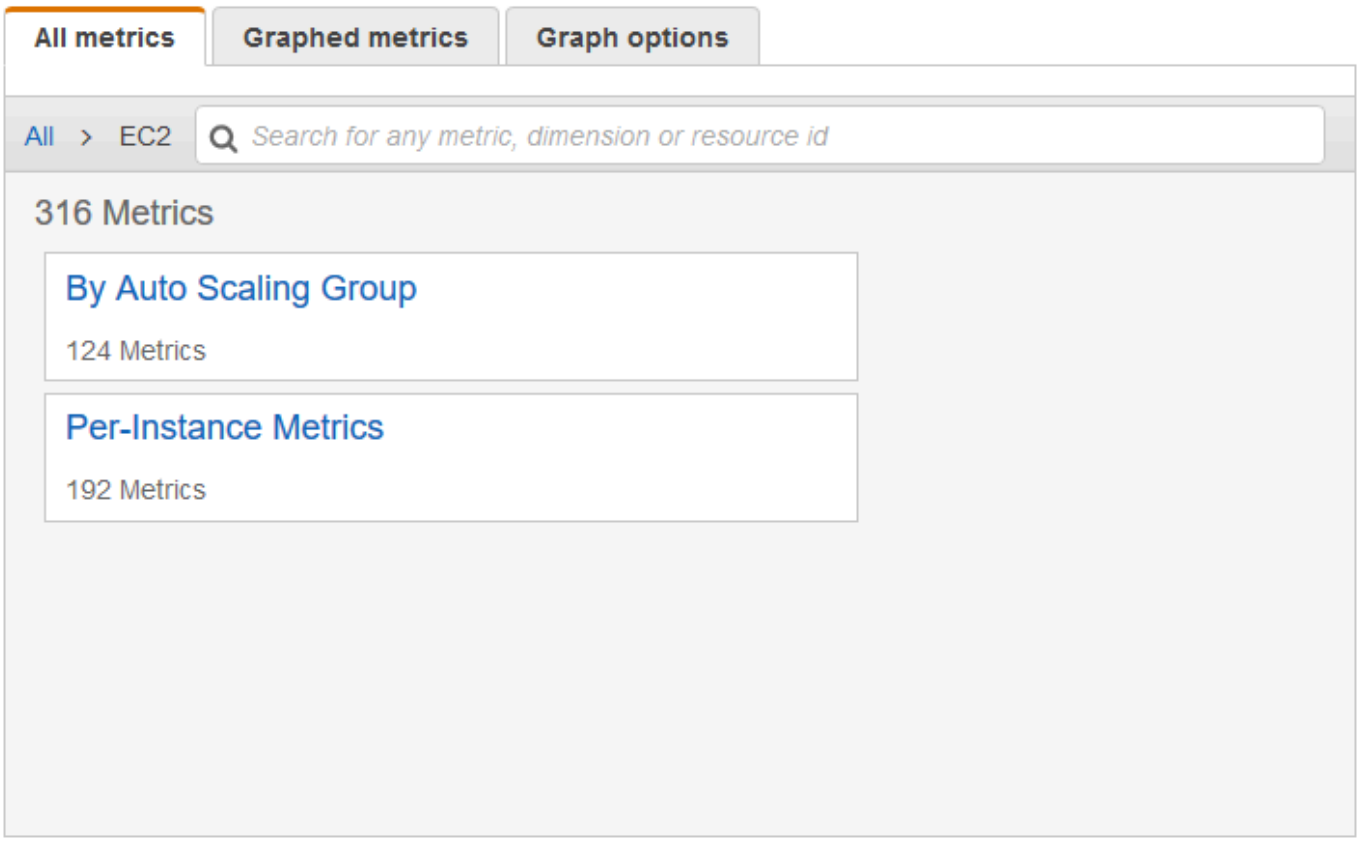
- インスタンスの ID が必要です。インスタンス ID は、Amazon EC2 コンソールまたは describe-instances コマンドを使って取得します。
- デフォルトでは、基本モニターリングが有効化されていますが、詳細モニターリングを有効化することもできます。詳細については、Linux インスタンス用の Amazon EC2 ユーザーガイドの「インスタンスの詳細モニターリングの有効化または無効化」を参照してください。

コンソールを使用して特定のインスタンスの平均 CPU 使用率を表示するには

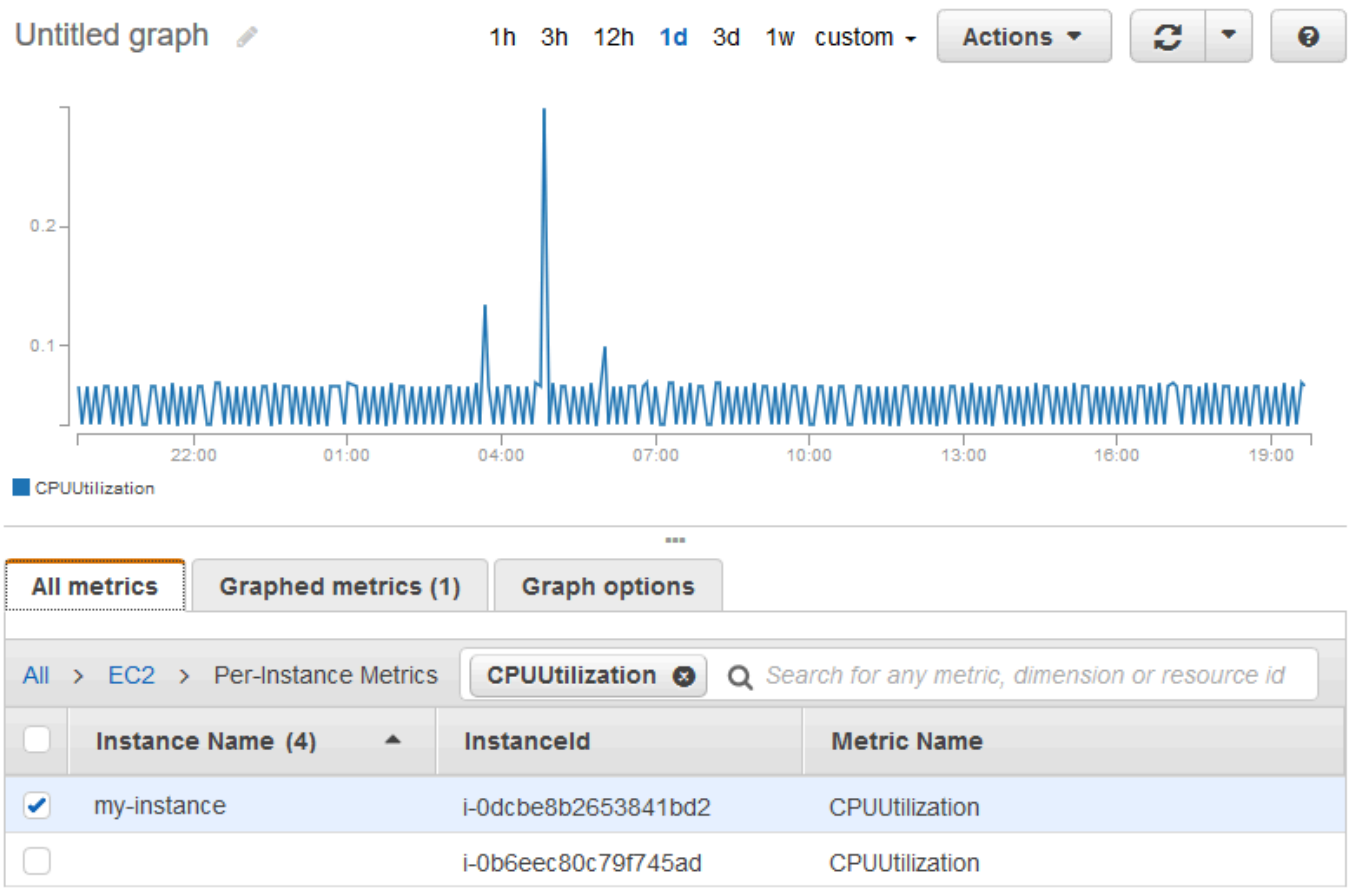
1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで [Metrics (メトリクス)] を選択します。
3. [EC2] メトリクス名前空間を選択します。



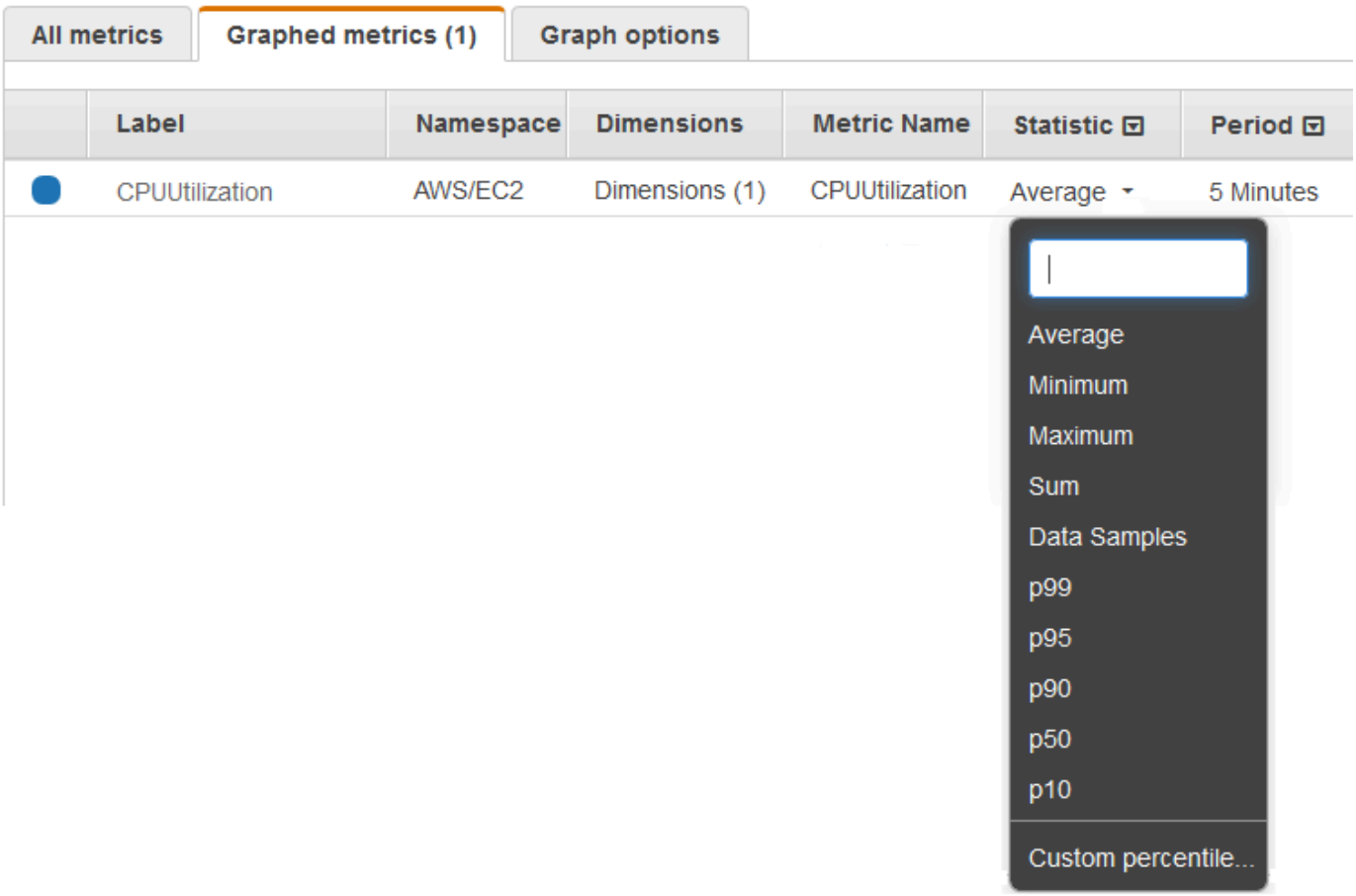
4. [インスタンス別メトリクス] のディメンションを選択します。



5. 検索フィールドに **CPUUtilization** と入力して Enter キーを押します。特定のインスタンスの行を選択します。これにより、そのインスタンスの [CPUUtilization] メトリクスのグラフが表示されます。グラフの名前を変更するには、鉛筆アイコンを選択します。時間範囲を変更するには、事前定義済みの値を選択するか、[custom] を選択します。



6. 統計を変更するには、[Graphed metrics] タブを選択します。列見出しまたは個々の値を選択し、統計または事前定義パーセンタイルのうち 1 つを選択するか、カスタムパーセンタイル (**p99.999** など) を指定します。



7. 期間を変更するには、[Graphed metrics] タブを選択します。列見出しまたは個々の値を選択し、次に異なる値を選択します。

AWS CLI を使用して、EC2 インスタンスあたりの CPU 使用率を取得するには

次のように get-metric-statistics コマンドを使用して、指定のインスタンスの CPUUtilization メトリクスを取得します。

```
aws cloudwatch get-metric-statistics --namespace AWS/EC2 --metric-name CPUUtilization \
--dimensions Name=InstanceId,Value=i-1234567890abcdef0 --statistics Maximum \
--start-time 2016-10-18T23:18:00 --end-time 2016-10-19T23:18:00 --period 360
```

返される統計は、リクエストされた 24 時間間隔の 6 分の値です。それぞれの値は、特定の 6 分間の間に指定されたインスタンスの最大 CPU 使用率を表しています。データポイントは時系列順に返されません。例に示した出力の先頭 (完全な出力には、6 分ごと 24 時間のデータポイントを含む) を以下に示します。

```
{
```

```
    "Datapoints": [
        {
            "Timestamp": "2016-10-19T00:18:00Z",
            "Maximum": 0.33000000000000002,
            "Unit": "Percent"
        },
        {
            "Timestamp": "2016-10-19T03:18:00Z",
            "Maximum": 99.670000000000002,
            "Unit": "Percent"
        },
        {
            "Timestamp": "2016-10-19T07:18:00Z",
            "Maximum": 0.34000000000000002,
            "Unit": "Percent"
        },
        ...
    ],
    "Label": "CPUUtilization"
}
```

統計をリソース間で集計する

複数のリソース間で AWS リソースのメトリクスを集計することができます。メトリクスは、リージョン間で完全に分離されていますが、メトリクス計算を使用して、リージョン間で類似したメトリクスを集計できます。詳細については、「Metric Math を使用する」を参照してください。

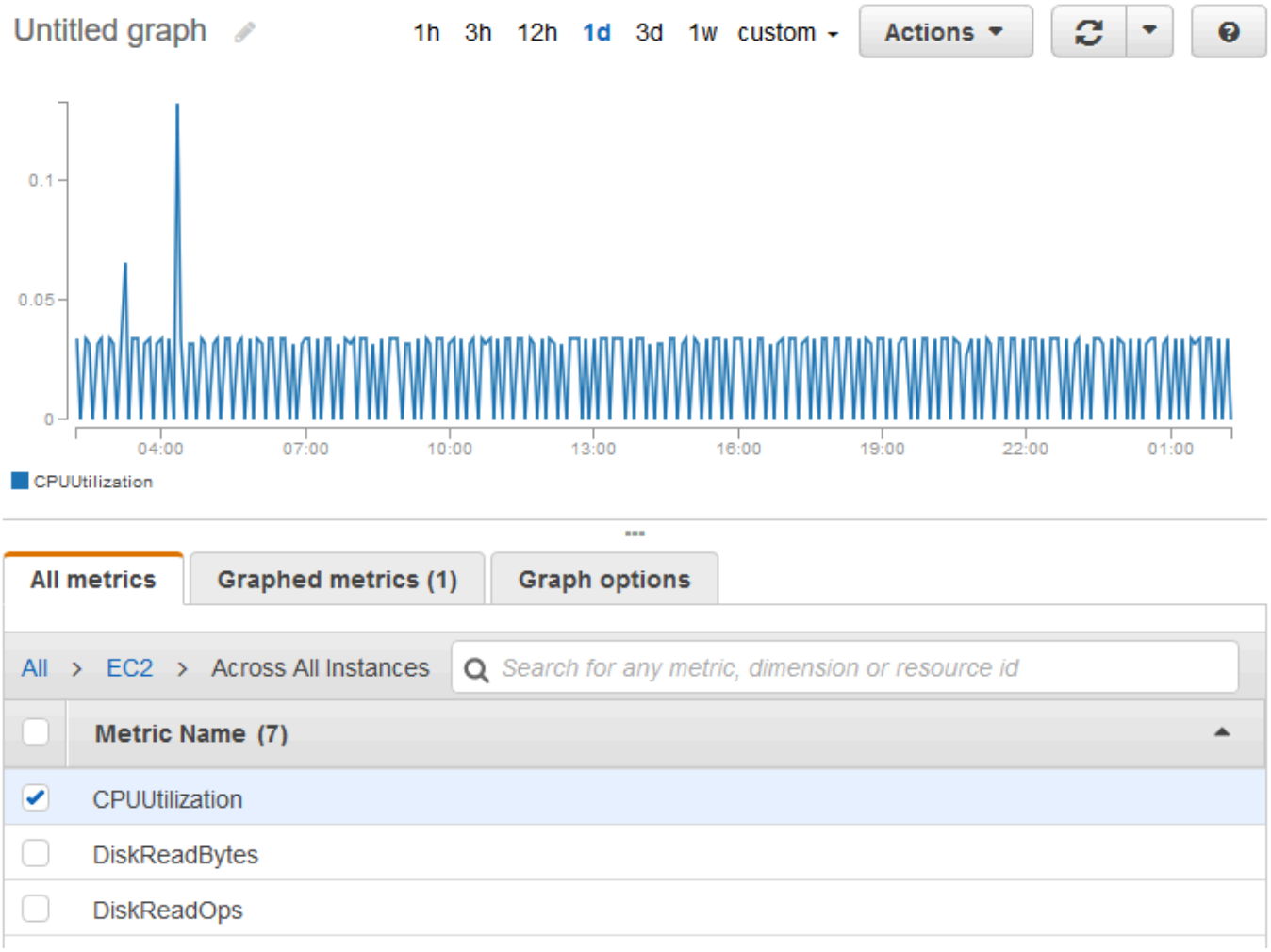
たとえば、詳細モニターリングが有効化されているすべての EC2 インスタンスで統計を集計することができます。その場合、基本モニターリングを使用するインスタンスは含まれません。そのため、1 分間隔でデータを提供する詳細モニターリング (有料) を有効化する必要があります。詳細については、Linux インスタンス用の Amazon EC2 ユーザーガイドの「インスタンスの詳細モニターリングの有効化または無効化」を参照してください。

この例では、EC2 インスタンスの平均 CPU 使用率を取得する方法を説明します。ディメンションを指定していないため、CloudWatch は、AWS/EC2 名前空間にある全ディメンションの統計を返します。その他のメトリクスの統計を取得するには、CloudWatch メトリクスを発行する AWS のサービス を参照してください。

> **Important**
>
> AWS 名前空間にあるすべてのディメンションを取得するこの手法は、Amazon CloudWatch に発行するカスタム名前空間では機能しません。カスタム名前空間の場合、データポイントを含む統計を取得するには、そのデータポイントに関連付けられたディメンション一式をすべて指定する必要があります。

EC2 インスタンスの平均 CPU 使用率を表示するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで [Metrics (メトリクス)] を選択します。
3. [EC2] 名前空間を選択し、次に [Across All Instances] を選択します。
4. [CPUUtilization] を含む行を選択します。すべての EC2 インスタンスのメトリクスがグラフとして表示されます。グラフの名前を変更するには、鉛筆アイコンを選択します。時間範囲を変更するには、事前定義済みの値を選択するか、[custom] を選択します。



5. 統計を変更するには、[Graphed metrics] タブを選択します。列見出しまたは個々の値を選択し、統計または事前定義パーセンタイルのうち 1 つを選択するか、カスタムパーセンタイル (**p95.45** など) を指定します。
6. 期間を変更するには、[Graphed metrics] タブを選択します。列見出しまたは個々の値を選択し、次に異なる値を選択します。

AWS CLI を使用して EC2 インスタンス間での平均で CPU 使用率を取得するには

以下のように get-metric-statistics コマンドを使用します。

```
aws cloudwatch get-metric-statistics --namespace AWS/EC2 --metric-name CPUUtilization
 --statistics "Average" "SampleCount" \
--start-time 2016-10-11T23:18:00 --end-time 2016-10-12T23:18:00 --period 3600
```

出力例を次に示します。

```
{
    "Datapoints": [
        {
            "SampleCount": 238.0,
            "Timestamp": "2016-10-12T07:18:00Z",
            "Average": 0.038235294117647062,
            "Unit": "Percent"
        },
        {
            "SampleCount": 240.0,
            "Timestamp": "2016-10-12T09:18:00Z",
            "Average": 0.16670833333333332,
            "Unit": "Percent"
        },
        {
            "SampleCount": 238.0,
            "Timestamp": "2016-10-11T23:18:00Z",
            "Average": 0.041596638655462197,
            "Unit": "Percent"
        },
        ...
    ],
    "Label": "CPUUtilization"
}
```


Auto Scaling グループ別に統計を集計する

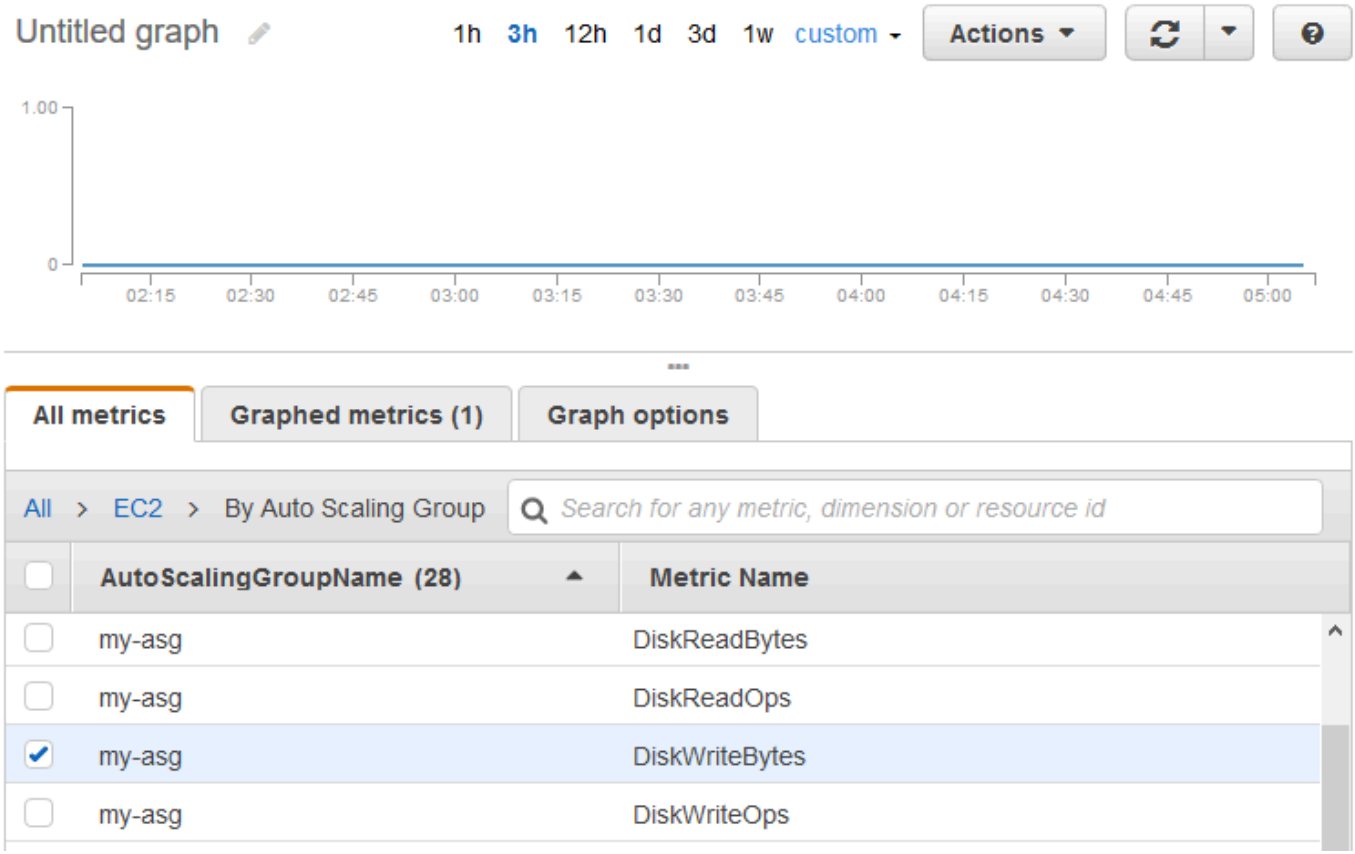
Auto Scaling グループ内で EC2 インスタンスの統計を集計することができます。メトリクスはリージョン別に完全に独立していますが、CloudWatch のメトリクス数式を使用すると、複数のリージョンのメトリクスを集計および変換できます。クロスアカウントダッシュボードを使用して、複数の異なるアカウントのメトリクスに対してメトリック数式を実行することもできます。

この例では、1 つの Auto Scaling グループについて、ディスクに書き込まれた総バイト数を取得する方法を説明します。合計は、指定された Auto Scaling グループのすべての EC2 インスタンスで、24 時間おきに 1 分間に対して算出されます。

コンソールを使い、Auto Scaling グループ内のインスタンスの DiskWriteBytes を表示するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで [Metrics (メトリクス)] を選択します。

3. [EC2] 名前空間を選択し、次に [By Auto Scaling Group] (Auto Scaling グループ別) を選択します。
4. [DiskWriteBytes] メトリクスの行と特定の Auto Scaling グループを選択します。すると、その Auto Scaling グループ内にあるインスタンスのメトリクスがグラフとして表示されます。グラフの名前を変更するには、鉛筆アイコンを選択します。時間範囲を変更するには、事前定義済みの値を選択するか、[custom] を選択します。



5. 統計を変更するには、[Graphed metrics] タブを選択します。列見出しまたは個々の値を選択し、統計または事前定義パーセンタイルのうち 1 つを選択するか、カスタムパーセンタイル (**p95.45** など) を指定します。
6. 期間を変更するには、[Graphed metrics] タブを選択します。列見出しまたは個々の値を選択し、次に異なる値を選択します。

AWS CLI を使用して Auto Scaling グループのインスタンス内の DiskWriteBytes を取得するには

以下のように get-metric-statistics コマンドを使用します。

```
aws cloudwatch get-metric-statistics --namespace AWS/EC2 --metric-name DiskWriteBytes
--dimensions Name=AutoScalingGroupName,Value=my-asg --statistics "Sum" "SampleCount" \
```

```
--start-time 2016-10-16T23:18:00 --end-time 2016-10-18T23:18:00 --period 360
```

以下は出力例です。

```
{
    "Datapoints": [
        {
            "SampleCount": 18.0,
            "Timestamp": "2016-10-19T21:36:00Z",
            "Sum": 0.0,
            "Unit": "Bytes"
        },
        {
            "SampleCount": 5.0,
            "Timestamp": "2016-10-19T21:42:00Z",
            "Sum": 0.0,
            "Unit": "Bytes"
        }
    ],
    "Label": "DiskWriteBytes"
}
```

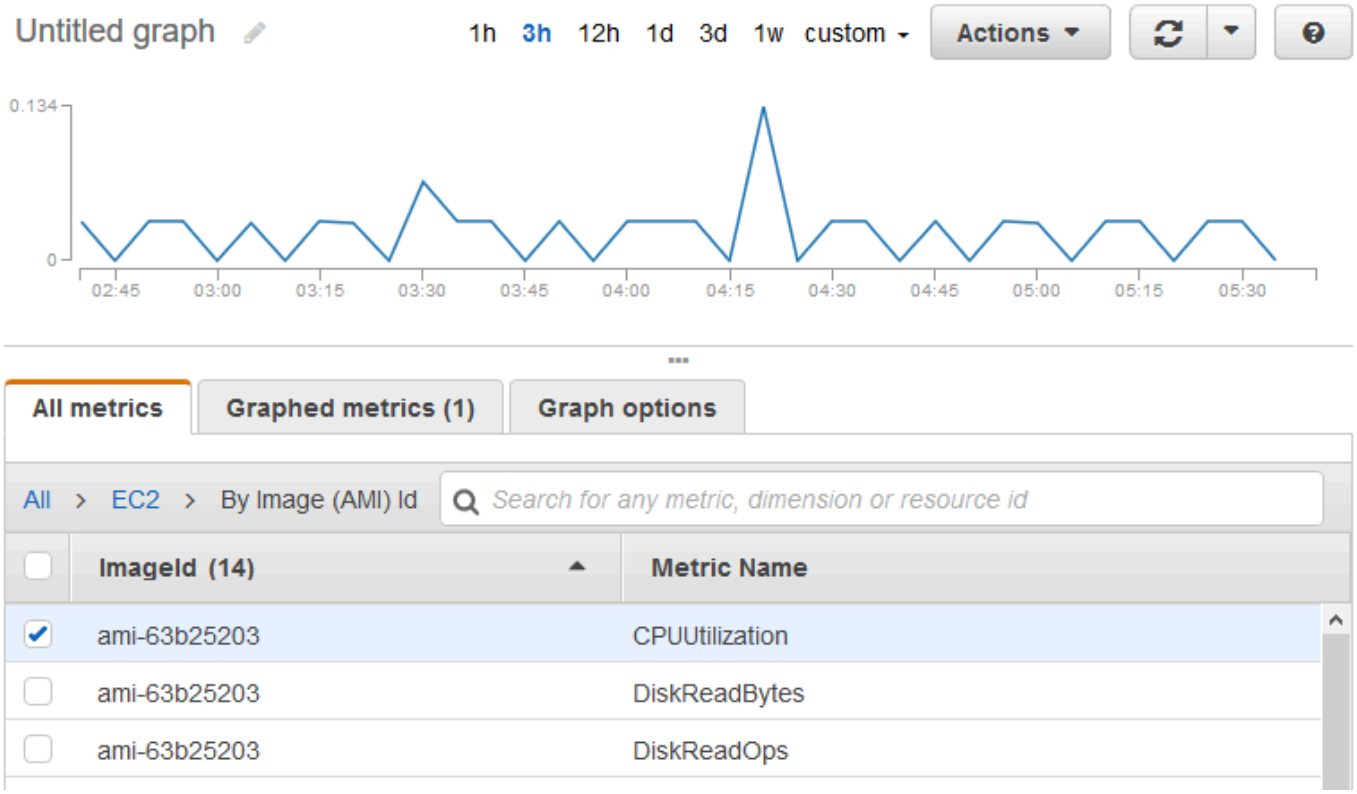
Amazon マシンイメージ (AMI) で統計を集計する

統計の集計は、詳細モニターリングが有効化されている EC2 インスタンスに対して行うことができます。その場合、基本モニターリングを使用するインスタンスは含まれません。詳細については、Linux インスタンス用の Amazon EC2 ユーザーガイドの「インスタンスの詳細モニターリングの有効化または無効化」を参照してください。

この例では、指定した AMI を使用するすべてのインスタンスの平均 CPU 使用率を特定する方法を説明します。平均値は、1 日間、60 秒間隔の平均値です。

コンソールを使い、AMI 別の平均 CPU 使用率を表示するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで [Metrics (メトリクス)] を選択します。
3. [EC2] 名前空間を選択し、次に [By Image (AMI) Id] を選択します。
4. [CPUUtilization] メトリクスの行と特定の AMI を選択します。すると、その AMI のメトリクスがグラフとして表示されます。グラフの名前を変更するには、鉛筆アイコンを選択します。時間範囲を変更するには、事前定義済みの値を選択するか、[custom] を選択します。



5. 統計を変更するには、[Graphed metrics] タブを選択します。列見出しまたは個々の値を選択し、統計または事前定義パーセンタイルのうち 1 つを選択するか、カスタムパーセンタイル (**p95.45** など) を指定します。
6. 期間を変更するには、[Graphed metrics] タブを選択します。列見出しまたは個々の値を選択し、次に異なる値を選択します。

AWS CLI を使用して、AMI 別の平均 CPU 使用率を取得するには

以下のように get-metric-statistics コマンドを使用します。

```
aws cloudwatch get-metric-statistics --namespace AWS/EC2 --metric-name CPUUtilization \
--dimensions Name=ImageId,Value=ami-3c47a355 --statistics Average \
--start-time 2016-10-10T00:00:00 --end-time 2016-10-11T00:00:00 --period 3600
```

オペレーションは、1 日間 1 時間おきの統計を返します。それぞれの値は、指定した AMI を実行する EC2 インスタンスの平均 CPU 使用率 (%) を表します。以下は出力例です。

```
{
    "Datapoints": [
        {
```

```
            "Timestamp": "2016-10-10T07:00:00Z",
            "Average": 0.041000000000000009,
            "Unit": "Percent"
        },
        {
            "Timestamp": "2016-10-10T14:00:00Z",
            "Average": 0.079579831932773085,
            "Unit": "Percent"
        },
        {
            "Timestamp": "2016-10-10T06:00:00Z",
            "Average": 0.036000000000000011,
            "Unit": "Percent"
        },
        ...
    ],
    "Label": "CPUUtilization"
}
```

カスタムメトリクスをパブリッシュする

AWS CLI または API を使用して、独自のメトリクスを CloudWatch に発行できます。AWS Management Console で発行したメトリクスの統計グラフを表示できます。

CloudWatch は、一連のデータポイントとしてメトリクスに関するデータを格納します。各データポイントには関連するタイムスタンプがあります。さらに、統計セットという集約されたデータポイント一式をパブリッシュすることもできます。

トピック

高解像度のメトリクス

各メトリクスは次のいずれかです。

- 詳細度が 1 分のデータを含む、標準の解像度
- 詳細度が 1 秒のデータを含む高解像度

AWS のサービスによって生成されたメトリクスは、デフォルトで標準解像度になります。カスタムメトリクスを発行するときは、標準解像度または高解像度のいずれかとして定義できます。高分解能のメトリクスをパブリッシュすると、CloudWatch はそれを 1 秒の分解能で保存します。ユーザーは、1 秒、5 秒、10 秒、30 秒、または 60 秒の倍数の期間でメトリクスを読み取り、取得できます。

高解像度メトリクスを使用すれば、アプリケーションの 1 分未満のアクティビティをより迅速に把握できます。PutMetricData がカスタムメトリクスを呼び出す場合、課金されることに注意してください。そのため、高解像度で PutMetricData を頻繁に呼び出すと、高額な料金が発生する可能性があります。CloudWatch の料金の詳細については、Amazon CloudWatch の料金をご覧ください。

高解像度メトリクスでアラームを設定する場合、10 秒または 30 秒の期間で高解像度アラームを指定するか、60 秒の倍数の期間で通常のアラームを設定できます。10 秒または 30 秒の期間の高解像度アラームでは、料金が高くなります。

ディメンションを使用する

カスタムメトリクスでは、--dimensions パラメータは一般的です。ディメンションでは、これに加えて、メトリクスの内容やメトリクスによって保存されるデータまで分かります。1 つのメトリクスには最大 30 個のディメンションを割り当てることができ、各ディメンションは名前と値のペアで定義されます。

使用するコマンドが異なる場合は、使用するディメンションも異なります。put-metric-data を使用する場合は、各ディメンションを *MyName*=*MyValue* と指定しますが、get-metric-statistics または put-metric-alarm を使用する場合は、Name=*MyName*、Value=*MyValue* を使用します。たとえば、次のコマンドでは、「InstanceId」と「InstanceType」という名前の 2 つのディメンションを持つ「Buffers」メトリクスを発行します。

```
aws cloudwatch put-metric-data --metric-name Buffers --namespace MyNameSpace --unit
 Bytes --value 231434333 --dimensions InstanceId=1-23456789,InstanceType=m1.small
```

このコマンドは、同一メトリクスの統計情報を取得します。ディメンションが 1 つの場合は、名前と値をカンマで区切り、複数ある場合は、1 つめのディメンションと 2 つめのディメンションの間にスペースを使用します。

```
aws cloudwatch get-metric-statistics --metric-name Buffers --namespace MyNameSpace --
dimensions Name=InstanceId,Value=1-23456789 Name=InstanceType,Value=m1.small --start-
time 2016-10-15T04:00:00Z --end-time 2016-10-19T07:00:00Z --statistics Average --period
 60
```

また、1 つのメトリクスに複数のディメンションを含む場合は、get-metric-statistics を使用するときに、定義されているディメンションごとに値を指定する必要があります。例えば、Amazon S3 メトリクス BucketSizeBytes に、BucketName と StorageType の 2 つのディメンションが含まれている場合は、get-metric-statistics を使用して両方のディメンションを指定する必要があります。

```
aws cloudwatch get-metric-statistics --metric-name BucketSizeBytes --start-time
 2017-01-23T14:23:00Z --end-time 2017-01-26T19:30:00Z --period 3600 --namespace
 AWS/S3 --statistics Maximum --dimensions Name=BucketName,Value=MyBucketName
 Name=StorageType,Value=StandardStorage --output table
```

メトリクスに定義されているディメンションを確認するには、list-metrics コマンドを使用します。

単一データポイントを公開する

新規または既存のメトリクスに単一のデータポイントをパブリッシュするには、1 つの値とタイムスタンプで put-metric-data コマンドを呼び出します。たとえば、次のアクションはそれぞれ 1 つのデータポイントを発行しています。

```
aws cloudwatch put-metric-data --metric-name PageViewCount --namespace MyService --
value 2 --timestamp 2016-10-20T12:00:00.000Z
aws cloudwatch put-metric-data --metric-name PageViewCount --namespace MyService --
value 4 --timestamp 2016-10-20T12:00:01.000Z
aws cloudwatch put-metric-data --metric-name PageViewCount --namespace MyService --
value 5 --timestamp 2016-10-20T12:00:02.000Z
```

新しいメトリクス名でこのコマンドを呼び出すと、CloudWatch がメトリクスを作成します。そうでない場合、CloudWatch は指定した既存のメトリクスとデータを関連付けます。

> **Note**
>
> メトリクスを作成したら、get-metric-statistics コマンドを用いてその新規メトリクスの統計を取得できるようになるまで最大 2 分かかります。ただし、list-metrics コマンドを用いて取得したメトリクスのリストに新規メトリクスが表示されるまでは最大 15 分かかることがあります。

粒度が 1 秒の 1,000 分の 1 のタイムスタンプでデータポイントをパブリッシュできますが、CloudWatch は、データを最小粒度の 1 分に集約します。CloudWatch は 1 分の期間ごとに受け取った値の平均 (すべての項目の合計を項目数で割った値) と、同じ期間内のサンプル数、最大値、最小値を記録します。たとえば、前の例の PageViewCount メトリクスには、3 つのデータポイントで、数秒違いのタイムスタンプがあります。期間が 1 分に設定されている場合、3 つのデータポイントはタイムスタンプがすべて 1 分の期間内にあるため、CloudWatch はそれらを集約します。

パブリッシュしたデータポイントを基に、get-metric-statistics コマンドを用いて統計を取得できます。

```
aws cloudwatch get-metric-statistics --namespace MyService --metric-name PageViewCount
 \
--statistics "Sum" "Maximum" "Minimum" "Average" "SampleCount" \
--start-time 2016-10-20T12:00:00.000Z --end-time 2016-10-20T12:05:00.000Z --period 60
```

以下は出力例です。

```
{
    "Datapoints": [
        {
            "SampleCount": 3.0,
            "Timestamp": "2016-10-20T12:00:00Z",
            "Average": 3.6666666666666665,
            "Maximum": 5.0,
            "Minimum": 2.0,
            "Sum": 11.0,
            "Unit": "None"
        }
    ],
    "Label": "PageViewCount"
}
```

統計セットを公開する

さらに、CloudWatch にパブリッシュする前にデータを集約することができます。各分に複数のデータポイントがある場合、データを集約して put-metric-data への呼び出し回数を最小化できます。たとえば、互いに 3 秒以内の位置にある 3 つのデータポイントに対して put-metric-data を複数回呼び出す代わりに、--statistic-values パラメータを使用して、1 回の呼び出しで発行できる統計セットにデータを集約できます。

```
aws cloudwatch put-metric-data --metric-name PageViewCount --namespace MyService
 --statistic-values Sum=11,Minimum=2,Maximum=5,SampleCount=3 --
timestamp 2016-10-14T12:00:00.000Z
```

CloudWatch は、raw データポイントを使用してパーセンタイルを計算します。統計セットを使用してデータを発行する場合は、以下の条件のいずれかが真である場合を除き、このデータのパーセンタイル統計を取得することはできません。

- 統計セットの SampleCount が 1
- 統計セットの Minimum と Maximum が同一である

値ゼロを公開する

データが散発的で、関連データがない期間がある場合、その期間に対して値ゼロ 0 をパブリッシュするか、全く値なしにするかを選択できます。アプリケーションの状態をモニターリングするために PutMetricData を周期的に呼び出す場合、値なしにするのではなく、値ゼロ (0) をパブリッシュすることができます。例えば、アプリケーションが 5 分ごとにメトリクスをパブリッシュできない場合、CloudWatch アラームを設定できます。そのようなアプリケーションに関連データがない期間ではゼロ (0) をパブリッシュさせることができます。

また、データポイントの総数を追跡する場合、または最小値や平均値などの統計に値 0 のデータポイントを含める場合も 0 をパブリッシュすることがあります。

メトリクスの公開を停止する

CloudWatch へのカスタムメトリクスの公開を停止するには、アプリケーションまたはサービスのコードを変更して、PutMetricData の使用を停止します。CloudWatch はアプリケーションからメトリクスをプルせず、プッシュされたもののみを受信するため、メトリクスの公開を停止するには、ソースでメトリクスを停止する必要があります。

Amazon CloudWatch でのアラームの使用

Amazon CloudWatch でメトリクスと複合アラームを作成できます。

- 1 つの CloudWatch メトリクスを監視する、または複数の CloudWatch メトリクスに基づく数式の結果を監視するメトリクスアラーム。アラームは、メトリクスや式の値が複数の期間にわたって特定のしきい値を超えた場合に 1 つ以上のアクションを実行します。アクションでは、Amazon SNS トピックに通知を送信したり、Amazon EC2 アクションまたは Amazon EC2 Auto Scaling アクションを実行できます。また、Systems Manager で OpsItem またはインシデントを作成できます。
- 複合アラームには、作成した他のアラームのアラーム状態を考慮したルール式が含まれます。複合アラームは、ルールのすべての条件が満たされた場合に限り、ALARM 状態になります。複合アラームのルール式で指定されたアラームには、メトリクスアラームやその他の複合アラームを含めることができます。

  複合アラームを使用すると、アラームノイズを低減できます。複数のメトリクスアラームを作成する、複合アラームを作成する、または複合アラームに対してのみアラートを設定することができます。たとえば、複合が ALARM 状態になるのは、基盤となるすべてのメトリクスアラームが ALARM 状態である場合だけです。

  複合アラームは、状態が変更されたときに Amazon SNS 通知を送信できます。また、ALARM 状態になったときに Systems Manager の OpsItems またはインシデントを作成できますが、EC2 アクションまたは Auto Scaling アクションを実行することはできません。

> **Note**
>
> AWS アカウントでは、必要な数だけアラームを作成できます。

ダッシュボードにアラームを追加すると、複数のリージョンにまたがる AWS リソースとアプリケーションを監視してアラートを受信できます。ダッシュボードにアラームを追加すると、アラームは、`INSUFFICIENT_DATA` 状態の場合はグレーに変わり、`ALARM` 状態の場合は赤に変わります。`OK` 状態の場合、アラームは色なしで表示されます。

また、CloudWatch コンソールのナビゲーションペインで、最近アクセスしたアラームを [Favorites and recents] (お気に入りと最近のアクセス) オプションからお気に入りに登録できます。[Favorites

and recents] (お気に入りと最近のアクセス) オプションには、お気に入りのアラームと最近アクセスしたアラームの列があります。

アラームは、アラームの状態が変わった場合にのみアクションを呼び出します。例外は、Auto Scaling アクションを持つアラームの場合です。Auto Scaling アクションの場合、アラームは 1 分間に 1 回アクションを呼び出し続け、アラームが新しい状態のままになります。

アラームは、同じアカウントのメトリックスを監視できます。CloudWatch コンソールでクロスアカウント機能を有効にしている場合は、他のAWSアカウントを監視するアラームも作成できます。クロスアカウント複合アラームの作成はサポートされていません。数式を使用するクロスアカウントアラームの作成がサポートされていますが、ANOMALY_DETECTION_BAND、INSIGHT_RULE、および SERVICE_QUOTA 関数は、クロスアカウントアラームではサポートされていません。

> **Note**
>
> CloudWatch は、指定したアクションのテストや検証は行いません。また、存在しないアクションの呼び出しから生じる Amazon EC2 Auto Scaling や Amazon SNS のエラーも検出しません。アラームアクションが存在していることを確認してください。

メトリクスアラームの状態

メトリクスアラームには次の状態があります。

- OK – メトリクスや式は、定義されているしきい値の範囲内です。
- ALARM – メトリクスまたは式が、定義されているしきい値を超えています。
- INSUFFICIENT_DATA – アラームが開始直後であるか、メトリクスが利用できないか、メトリクス用のデータが不足しているため、アラームの状態を判定できません。

アラームの評価

アラームを作成するときに、CloudWatch がアラームの状態を変更するタイミングを評価できるように 3 つの設定を指定します。

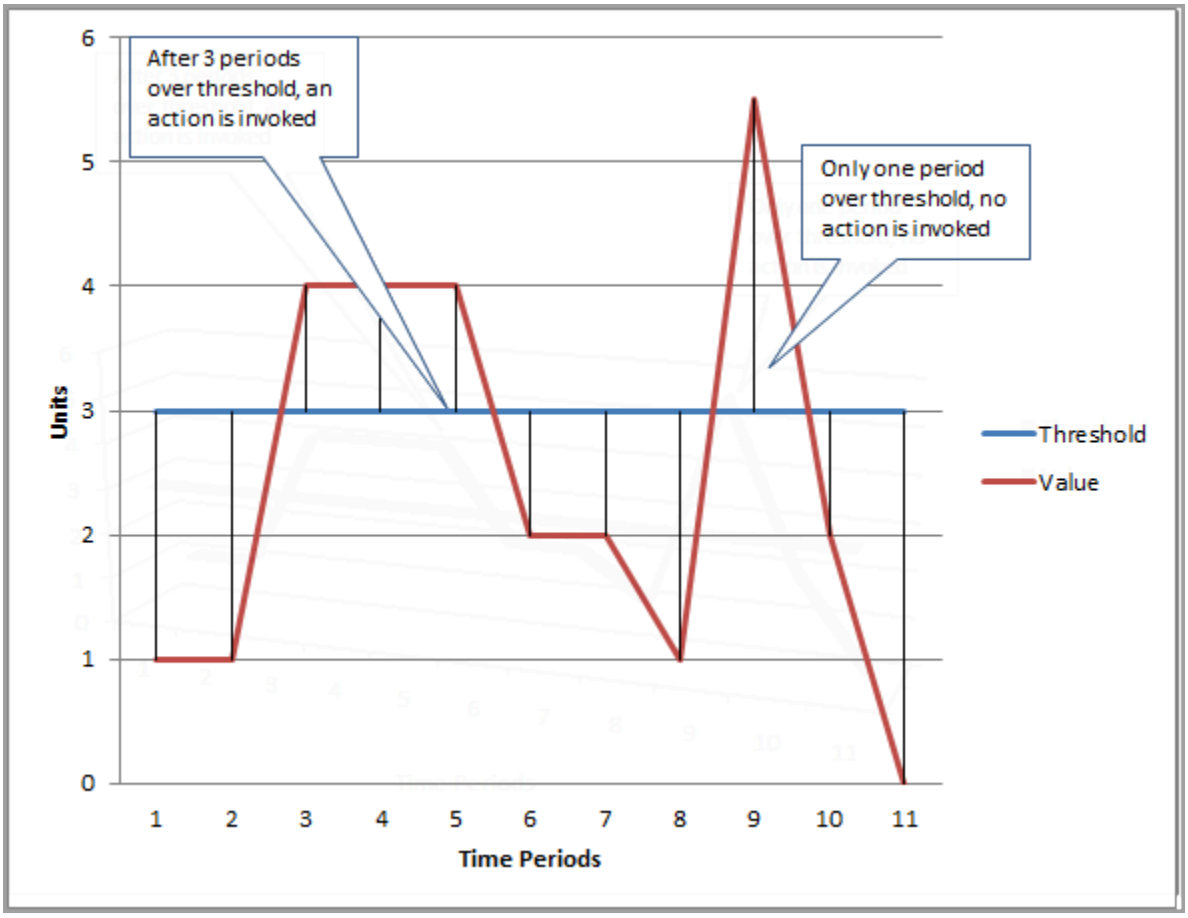
- [期間] は、アラームの各データポイントを作成することを目的として、メトリクスや式を評価するために使用する期間です。これは秒単位で表されます。
- [Evaluation Periods (評価期間)] は、アラームの状態を決定するまでに要する最新の期間またはデータポイントの数です。

- [Datapoints to Alarm (アラームを実行するデータポイント)] は、アラームが ALARM 状態に移るためにしきい値を超過する必要がある評価期間内のデータポイントの数です。しきい値を超過したデータポイントは連続している必要はありません。しかしすべてが [Evaluation Period] (評価期間) に相当する直近のデータポイント数に含まれている必要があります。

期間が 1 分以上の場合、アラームは 1 分ごとに評価され、評価は [期間] と [評価期間] で定義されている時間枠に基づいて行われます。例えば、[期間] が 5 分 (300 秒) で、[評価期間] が 1 の場合、5 分終了時に、アラームは 1 分 ~ 5 分のデータに基づいて評価されます。続いて、6 分終了時に、2 分 ~ 6 分のデータに基づいてアラームが評価されます。

アラーム期間が 10 秒または 30 秒の場合、アラームは 10 秒ごとに評価されます。

次の図では、メトリクスアラームのアラームしきい値が 3 単位に設定されています。[Evaluation Periods (評価期間)] と [Datapoints to Alarm (アラームを実行するデータポイント)] はどちらも 3 です。つまり、連続する最新の 3 つの期間内の既存のデータポイントすべてがしきい値を上回ると、アラームは ALARM 状態になります。この図では、第 3 期間から第 5 期間にかけてこれが発生します。第 6 期間で値がしきい値を下回り、評価対象の期間の 1 つが超過していないため、アラームの状態は OK に変化します。第 9 期間中のしきい値に再度超過がありますが、1 つの期間のみです。そのため、アラームの状態は OK のままです。



[Evaluation Periods (評価期間)] と [Datapoints to Alarm (アラームを実行するデータポイント)] に異なる値を設定する場合、「N 個中 M 個」のアラームを設定することになります。[Datapoints to Alarm (アラームを実行するデータポイント)] が (「M」)、[Evaluation Periods (評価期間)] が (「N」) です。評価間隔は、評価期間の数に期間の長さを乗じて算出されます。たとえば、1 分の期間を持つ 5 個のデータポイントのうち 4 個を設定した場合、評価間隔は 5 分です。10 分の期間を持つ 3 個のデータポイントのうち 3 個を設定した場合、評価間隔は 30 分です。

> **Note**
>
> アラームの作成後すぐにデータポイントが欠落し、アラームの作成前にメトリクスが CloudWatch に報告されている場合、CloudWatch はアラームを評価する際にアラーム作成前の直近のデータポイントを取得します。

アラームアクション

OK、ALARM、および INSUFFICIENT_DATA の状態の間で状態が変わったときに、アラームが実行するアクションを指定できます。

3 つの状態それぞれに移行するほとんどのアクションを設定できます。Auto Scaling アクションを除き、アクションは状態遷移時にのみ発生し、その状態が数時間または数日間続く場合は再実行されません。1 つのアラームに対して複数のアクションが許可されているという事実を利用して、しきい値を超えたときと、しきい値を超えた状態が終了したときに E メールを送信することができます。これにより、スケーリングまたは回復のアクションが予想どおりにトリガーされ、期待どおりに機能していることを確認できます。

以下がアラームアクションとしてサポートされています。

- [Amazon Simple Notification Service トピックを使用] して、1 つ以上のサブスクライバーに通知します。サブスクライバーはアプリケーションでも個人でもかまいません。Amazon SNS の詳細については、「Amazon SNS とは」を参照してください。
- [Lambda 関数を呼び出す]。これは、アラーム状態の変化に対するカスタムアクションを自動化する最も簡単な方法です。
- EC2 メトリクスに基づくアラームは、EC2 インスタンスの停止、終了、再起動、復旧などの [EC2 アクションを実行] することもできます。詳細については、「EC2 インスタンスを停止、終了、再起動、または復旧するアラームを作成する」を参照してください。
- アラームは、[Auto Scaling グループをスケールする] アクションを実行することもできます。詳細については、「Amazon EC2 Auto Scaling のステップおよびシンプルスケーリングポリシー」を参照してください。
- アラームは、[Systems Manager Ops Center で OpsItems を作成するか、AWS Systems Manager Incident Manager でインシデントを作成する] ことができます。これらのアクションは、アラームが ALARM 状態になった場合にのみ実行されます。詳細については、「アラームから OpsItems を作成するように CloudWatch を設定する」および「インシデントの作成」を参照してください。

Lambda アラームアクション

CloudWatch アラームは、次の場合を除き、特定の状態変化に対して Lambda 関数の非同期呼び出しを保証します。

- 関数が存在しない場合。
- CloudWatch が Lambda 関数を呼び出す権限がない場合。

CloudWatch が Lambda サービスに到達できない場合、または別の理由でメッセージが拒否された場合、CloudWatch は呼び出しが成功するまで再試行します。Lambda はメッセージをキューに入れ、実行の再試行を処理します。Lambda が エラーを処理する方法など、この実行モデルの詳細については、「AWS Lambda デベロッパーガイド」の「非同期呼び出し」を参照してください。

Lambda 関数は、同じアカウントまたは他の AWS アカウントで呼び出すことができます。

アラームアクションとして Lambda 関数を呼び出すアラームを指定する場合、関数名、関数エイリアス、または関数の特定のバージョンを指定することもできます。

アラームアクションとして Lambda 関数を指定する場合は、関数のリソースポリシーを作成して、CloudWatch サービスプリンシパルによる関数の呼び出しを許可する必要があります。

これを行う 1 つの方法は、次の例のように AWS CLI を使用することです。

```
aws lambda add-permission \
--function-name my-function-name \
--statement-id AlarmAction \
--action 'lambda:InvokeFunction' \
--principal lambda.alarms.cloudwatch.amazonaws.com \
--source-account 111122223333 \
--source-arn arn:aws:cloudwatch:us-east-1:111122223333:alarm:alarm-name
```

または、次の例のいずれかに似たポリシーを作成し、それを関数に割り当てることもできます。

次の例では、アラームが配置されているアカウントを指定して、そのアカウント (111122223333) のアラームだけが関数を呼び出せるようにします。

```
{
    "Version": "2012-10-17",
    "Id": "default",
    "Statement": [{
        "Sid": "AlarmAction",
        "Effect": "Allow",
        "Principal": {
            "Service": "lambda.alarms.cloudwatch.amazonaws.com"
        },
        "Action": "lambda:InvokeFunction",
        "Resource": "arn:aws:lambda:us-east-1:444455556666:function:function-name",
        "Condition": {
            "StringEquals": {
                "AWS:SourceAccount": "111122223333"
            }
```

```
        }
    }]
}
```

次の例では対象範囲が狭いため、指定したアカウントの指定したアラームだけが関数を呼び出せるようになっています。

```
{
  "Version": "2012-10-17",
  "Id": "default",
  "Statement": [
    {
      "Sid": "AlarmAction",
      "Effect": "Allow",
      "Principal": {
        "Service": "lambda.alarms.cloudwatch.amazonaws.com"
      },
      "Action": "lambda:InvokeFunction",
      "Resource": "arn:aws:lambda:us-east-1:444455556666:function:function-name",
      "Condition": {
        "StringEquals": {
          "AWS:SourceAccount": "111122223333",
          "AWS:SourceArn": "arn:aws:cloudwatch:us-east-1:111122223333:alarm:alarm-name"
        }
      }
    }]
}
```

ソースアカウントを指定しないポリシーは、混乱した代理問題に対して脆弱であるため、作成することはお勧めしません。

CloudWatch から Lambda に送信されたイベントオブジェクト

Lambda 関数をアラームアクションとして設定すると、CloudWatch は Lambda 関数を呼び出したときに JSON ペイロードを Lambda 関数に配信します。この JSON ペイロードは、関数のイベントオブジェクトとして機能します。この JSON オブジェクトからデータを抽出して、関数で使用できます。以下は、メトリクスアラームのイベントオブジェクトの例です。

```
{
  'source': 'aws.cloudwatch',
  'alarmArn': 'arn:aws:cloudwatch:us-east-1:444455556666:alarm:lambda-demo-metric-
alarm',
```

```
  'accountId': '444455556666',
  'time': '2023-08-04T12:36:15.490+0000',
  'region': 'us-east-1',
  'alarmData': {
    'alarmName': 'lambda-demo-metric-alarm',
    'state': {
      'value': 'ALARM',
      'reason': 'test',
      'timestamp': '2023-08-04T12:36:15.490+0000'
    },
    'previousState': {
      'value': 'INSUFFICIENT_DATA',
      'reason': 'Insufficient Data: 5 datapoints were unknown.',
      'reasonData':
 '{"version":"1.0","queryDate":"2023-08-04T12:31:29.591+0000","statistic":"Average","period":60
[],"threshold":5.0,"evaluatedDatapoints":[{"timestamp":"2023-08-04T12:30:00.000+0000"},
{"timestamp":"2023-08-04T12:29:00.000+0000"},
{"timestamp":"2023-08-04T12:28:00.000+0000"},
{"timestamp":"2023-08-04T12:27:00.000+0000"},
{"timestamp":"2023-08-04T12:26:00.000+0000"}]}',
      'timestamp': '2023-08-04T12:31:29.595+0000'
    },
    'configuration': {
      'description': 'Metric Alarm to test Lambda actions',
      'metrics': [
        {
          'id': '1234e046-06f0-a3da-9534-EXAMPLEe4c',
          'metricStat': {
            'metric': {
              'namespace': 'AWS/Logs',
              'name': 'CallCount',
              'dimensions': {
                'InstanceId': 'i-12345678'
              }
            },
            'period': 60,
            'stat': 'Average',
            'unit': 'Percent'
          },
          'returnData': True
        }
      ]
    }
  }
```

```
}
```

以下は、複合アラームのイベントオブジェクトの例です。

```
{
  'source': 'aws.cloudwatch',
  'alarmArn': 'arn:aws:cloudwatch:us-east-1:111122223333:alarm:SuppressionDemo.Main',
  'accountId': '111122223333',
  'time': '2023-08-04T12:56:46.138+0000',
  'region': 'us-east-1',
  'alarmData': {
    'alarmName': 'CompositeDemo.Main',
    'state': {
      'value': 'ALARM',
      'reason': 'arn:aws:cloudwatch:us-
east-1:111122223333:alarm:CompositeDemo.FirstChild transitioned to ALARM at Friday 04
 August, 2023 12:54:46 UTC',
      'reasonData': '{"triggeringAlarms":[{"arn":"arn:aws:cloudwatch:us-
east-1:111122223333:alarm:CompositeDemo.FirstChild","state":
{"value":"ALARM","timestamp":"2023-08-04T12:54:46.138+0000"}}]}',
      'timestamp': '2023-08-04T12:56:46.138+0000'
    },
    'previousState': {
      'value': 'ALARM',
      'reason': 'arn:aws:cloudwatch:us-
east-1:111122223333:alarm:CompositeDemo.FirstChild transitioned to ALARM at Friday 04
 August, 2023 12:54:46 UTC',
      'reasonData': '{"triggeringAlarms":[{"arn":"arn:aws:cloudwatch:us-
east-1:111122223333:alarm:CompositeDemo.FirstChild","state":
{"value":"ALARM","timestamp":"2023-08-04T12:54:46.138+0000"}}]}',
      'timestamp': '2023-08-04T12:54:46.138+0000',
      'actionsSuppressedBy': 'WaitPeriod',
      'actionsSuppressedReason': 'Actions suppressed by WaitPeriod'
    },
    'configuration': {
      'alarmRule': 'ALARM(CompositeDemo.FirstChild) OR
 ALARM(CompositeDemo.SecondChild)',
      'actionsSuppressor': 'CompositeDemo.ActionsSuppressor',
      'actionsSuppressorWaitPeriod': 120,
      'actionsSuppressorExtensionPeriod': 180
    }
  }
}
```

CloudWatch アラームの欠落データの処理の設定

場合によっては、メトリクスの予想されるすべてのデータポイントが CloudWatch にレポートされないことがあります。たとえば、接続が失われた場合、サーバーがダウンした場合、メトリクスがデータを断続的にのみ報告する設計になっている場合などに、これが発生する可能性があります。

CloudWatch では、アラームを評価する際の欠落データポイントの処理方法を指定できます。これにより、モニターリング対象のデータのタイプに適した場合にのみ ALARM 状態になるように、アラームを設定できます。欠落データが問題を示すものではない場合の誤検出を避けることができます。

各アラームが常に 3 つの状態のいずれかであるように、CloudWatch にレポートされるデータポイントはそれぞれ、次の 3 つのカテゴリのいずれかの状態に該当します。

- Not breaching (しきい値内)
- Breaching (しきい値を超過)
- Missing (見つからない)

アラームごとに、CloudWatch が欠落データポイントを次のいずれかとして処理するように指定できます。

- `notBreaching` – 欠落データポイントは「良好」かつしきい値内として扱われます。
- `breaching` – 欠落データポイントは「不良」とされ、しきい値超過として扱われます。
- `ignore` – 現在のアラーム状態が維持されます。
- `missing` – アラーム評価範囲内のすべてのデータポイントがない場合、アラームは INSUFFICIENT_DATA に移行します。

最適な選択は、メトリクスの種類とアラームの目的によって異なります。例えば、継続的にデータをレポートするメトリクスを使用してアプリケーションのロールバックアラームを作成する場合、欠落しているデータポイントは何らかの問題の存在を示している可能性があるため、違反として扱うことが推奨されます。ただし、Amazon DynamoDB の `ThrottledRequests` など、エラー発生時のみデータポイントを生成するメトリクスの場合は、`notBreaching` として欠落データを処理します。デフォルトの動作は `missing` です。

> **Important**
>
> Amazon EC2 メトリクスに設定されたアラームは、メトリクスデータポイントが欠落している場合、一時的に INSUFFICIENT_DATA 状態になることがあります。まれに、Amazon EC2

インスタンスが正常であっても、メトリクスレポートが中断されたときに、これが発生することがあります。アクションの停止、終了、再起動、復旧を行うように設定された Amazon EC2 メトリクスのアラームについては、欠落データを missing として扱い、アラームが ALARM 状態にある場合にのみトリガーされるように、これらのアラームを設定することが推奨されます。

アラームに最適なオプションを選択することで、アラームが、不要で誤解を招く状態に変更されることを防ぐだけでなく、システムの状態をさらに正確に表すことができます。

> **Important**
>
> AWS/DynamoDB 名前空間のメトリクスを評価するアラームでは、アラームで欠落データを扱う方法について他のオプションを選択した場合でも、欠落データは常に無視されます。AWS/DynamoDB メトリクスに欠落データがある場合、そのメトリクスを評価するアラームは現在の状態のままです。

データが欠落した場合のアラーム状態の評価方法

アラームが状態を変更するかどうかを評価するたびに、CloudWatch は [Evaluation Periods (評価期間)] に指定されている数よりも多くのデータポイントを取得しようとします。取得を試みるデータポイントの正確な数は、アラーム期間の長さと、基づいているメトリクスが標準解像度か高解像度かによって異なります。取得を試みるデータポイントのタイムフレームは評価範囲です。

CloudWatch がこれらのデータポイントを取得すると、次の処理が実行されます。

- 評価範囲内のデータポイントが欠落していない場合、CloudWatch は収集された最新のデータポイントに基づいてアラームを評価します。評価されるデータポイントの数は、アラームの [Evaluation Periods (評価期間)] と同じです。評価範囲内のよりさかのぼった時点からの余分なデータポイントは必要なく、無視されます。
- 評価範囲内のデータポイントの一部が欠落しているが、評価範囲から正常に取得された既存のデータポイントの合計数がアラームの [Evaluation Periods (評価期間)] 以上である場合、CloudWatch は、最新の正常に取得された最新のデータポイントに基づいたアラームの状態を評価します (評価範囲内のよりさかのぼった時点からの必要な追加データポイントを含む)。この場合、欠落データを処理する方法に設定した値は不要であり、無視されます。
- 評価範囲のデータポイントの一部が欠落しており、取得された既存のデータポイントの数がアラームの [Evaluation Periods (評価期間)] の数を下回る場合、CloudWatch によって、欠落データ部分

に欠落データの処理方法に指定された結果が入力され、アラームが評価されます。ただし、評価範囲内のすべての実際のデータポイントが評価に含まれます。CloudWatch は、欠落データポイントの使用を最小限に抑えます。

> **Note**
>
> この動作の特殊なケースは、メトリクスのフローが停止した後の一定期間、CloudWatch アラームが最後のデータポイントのセットを繰り返し再評価する可能性があることです。この再評価により、メトリクスのストリームが停止する直前にアラームの状態が変更されていた場合に、アクションが再実行される可能性があります。この動作を軽減するには、より短い期間を使用します。

次の表は、アラーム評価の動作の例を示しています。最初の表では、[Datapoints to Alarm (アラームを発生させるデータポイント数)] と [Evaluation Periods (評価期間)] はどちらも 3 です。CloudWatch は、直近の 3 個のデータポイントの一部が欠落している場合、アラームを評価する際に直近のデータポイントを 5 個取得します。5 はアラームの評価範囲です。

列 1 は、評価範囲が 5 であるため、最新の 5 つのデータポイントを示しています。これらのデータポイントが、右側が最新のデータポイントになるように表示されます。0 は違反していないデータポイント、X は違反しているデータポイント、- は欠落しているデータポイントを示します。

列 2 は、3 つの必要なデータポイントの欠落数を示します。最新の 5 個のデータポイントが評価されている場合でも、アラーム状態を評価する上で必要なデータポイントは 3 個 ([評価期間] の設定) のみです。列 2 のデータポイントの数は、欠落データの処理方法に関する設定を使用して「入力する」必要があるデータポイント数です。

列 3~6 では、列ヘッダーが欠落データの処理方法を示す値になります。これらの列の行には、欠損データを処理するためのこれらの可能な方法ごとに設定されたアラーム状態が表示されます。

データポイント	埋める必要のあるデータポイントの数	MISSING	IGNORE	BREACHING	NOT BREACHING
0 - X - X	0	OK	OK	OK	OK
- - - - 0	2	OK	OK	OK	OK

データポイント	埋める必要のあるデータポイントの数	MISSING	IGNORE	BREACHING	NOT BREACHING
- - - - -	3	INSUFFICIENT_DATA	現在の状態を維持	ALARM	OK
0 X X - X	0	ALARM	ALARM	ALARM	ALARM
- - X - -	2	ALARM	現在の状態を維持	ALARM	OK

前の表の 2 行目で、欠落データが超過として処理されても、アラームは OK のままです。既存のデータポイント 1 個が超過しておらず、これが超過として処理される欠落データポイント 2 個とともに評価されるためです。次回このアラームが評価される際にデータがまだ欠落したままであれば ALARM に遷移します。これは、超過していないデータポイントが、評価範囲に含まれなくなるためです。

3 番目の行では、最新の 5 つのデータポイントがすべて欠落しており、欠落したデータの処理方法に関するさまざまな設定がアラームの状態にどのように影響するかを示しています。欠落しているデータポイントが超過していると見なされるとアラームは ALARM 状態になり、超過していないと見なされるとアラームは OK 状態になります。欠落しているデータポイントを無視すると、アラームは欠落しているデータポイントよりも前の現在の状態を保持します。また、欠落しているデータポイントが欠落しているとみなされた場合、アラームには評価を行うのに十分な最近の実際のデータがないため、INSUFFICIENT_DATA に入ります。

4 行目では、最新の 3 つのデータポイントが超過しており、アラームの [Evaluation Periods (評価期間)] と [Datapoints to Alarm (アラームを実行するデータポイント)] が両方とも 3 に設定されているため、アラーム は ALARM 状態になります。この場合、欠落データポイントは無視され、欠落データの評価方法の設定は必要ありません。これは、評価する実際のデータポイントが 3 つあるためです。

行 5 は、早期アラーム状態と呼ばれる、アラーム評価の特別なケースを表します。詳細については、「アラーム状態への早期移行の回避」を参照してください。

次の表では、[期間] は再び 5 分に設定され、[Datapoints to Alarm (アラームを発生させるデータポイント数)] は 2 のみですが [評価期間] は 3 です。つまり 3 個のうち 2 個、N 個中 M のアラームです。

評価範囲は 5 です。これは、取得される最近のデータポイントの最大数であり、一部のデータポイントが欠落している場合に使用できます。

データポイント	欠落データポイント数	MISSING	IGNORE	BREACHING	NOT BREACHING
0 - X - X	0	ALARM	ALARM	ALARM	ALARM
0 0 X 0 X	0	ALARM	ALARM	ALARM	ALARM
0 - X - -	1	OK	OK	ALARM	OK
- - - - 0	2	OK	OK	ALARM	OK
- - - X -	2	ALARM	現在の状態を維持	ALARM	OK

行 1 と 2 では、最新の 3 つのデータポイントのうちの 2 つが超過しているため、アラームは常に ALARM 状態になります。行 2 では、評価範囲内の最も古い 2 つのデータポイントは不要です。これは、最新の 3 つのデータポイントのいずれも欠落していないためです。したがって、これらの 2 つの古いデータポイントは無視されます。

行 3 と 4 では、アラームは ALARM 状態になります。この場合、欠落データが超過として扱われる場合のみ、最新の 2 つの欠落データポイントは両方とも超過として扱われます。行 4 では、超過として扱われるこれらの 2 つの欠落データポイントは、ALARM 状態をトリガーするのに必要な 2 つの超過データポイントを提供します。

行 5 は、早期アラーム状態と呼ばれる、アラーム評価の特別なケースを表します。詳細については、以下のセクションを参照してください。

アラーム状態への早期移行の回避

CloudWatch アラーム評価には、データが断続的な場合にアラームが早く ALARM 状態になる誤アラームを回避しようとするロジックが含まれています。前のセクションの表の行 5 に示した例は、このロジックを示しています。これらの行および次の例では、[Evaluation Periods (評価期間)] は 3 で、評価範囲は 5 データポイントです。[Datapoints to Alarm (アラームを実行するデータポイント)] は 3 ですが、N 個のうち M の例を除いて、[Datapoints to Alarm (アラームを実行するデータポイント)] は 2 です。

アラームの最新のデータが - - - - X で、4 つのデータポイントが欠落しており、最新のデータポイントとして超過データポイントがあるとします。次のデータポイントは超過していない可能性が

あるため、データが - - - - X または - - - X - で、[Datapoints to Alarm (アラームを実行するデータポイント)] が 3 の場合、アラームはすぐに ALARM 状態になりません。このようにして、次のデータポイントが超過しておらず、データが - - - X 0 または - - X - 0 になる場合に誤検出が回避されます。

ただし、最後の数個のデータポイントが - - X - - の場合、欠落しているデータポイントが欠落していると見なされても、アラームは ALARM 状態になります。これは、[Evaluation Periods] (評価期間) 中のデータポイントのうち、利用可能な最も古い超過データポイントが、[Datapoints to Alarm] (アラームへのデータポイント) の値と少なくとも同じように古く、他のすべての最新のデータポイントが超過または欠落している場合にアラームが常に ALARM 状態になるように設計されているためです。この場合、使用可能なデータポイントの総数が M ([Datapoints to Alarm (アラームを実行するデータポイント)]) より少ない場合でも、アラームは ALARM 状態になります。

このアラームロジックは、N 個中 M 個のアラームにも適用されます。評価範囲で最も古い違反データポイントが [アラームを実行するデータポイント] の値と少なくとも同じくらい古く、最近のすべてのデータポイントが違反または欠落している場合、M ([アラームを実行するデータポイント]) の値に関係なくアラームは ALARM 状態になります。

高解像度アラーム

高解像度メトリクスでアラームを設定する場合、10 秒または 30 秒の期間で高解像度アラームを指定するか、60 秒の倍数の期間で通常のアラームを設定できます。高解像度のアラームには高い料金が発生します。高解像度メトリクスの詳細については、「カスタムメトリクスをパブリッシュする」を参照してください。

数式に基づくアラーム

1 つ以上の CloudWatch メトリクスに含む数式の結果に基づいてアラームを設定できます。アラーム用の数式には 10 個までのメトリクスを含めることができます。各メトリクスは同じ期間を使用している必要があります。

数式に基づくアラームの場合、データポイントの欠落を CloudWatch で処理する方法を指定できます。この場合、数式がそのデータポイントの値を返さない場合、そのデータポイントは欠落していると見なされます。

数式に基づくアラームでは Amazon EC2 アクションを実行できません。

メトリクスの数式と構文の詳細については、「Metric Math を使用する」を参照してください。

パーセンタイルベースの CloudWatch アラームおよび少数のデータサンプル

アラームの統計としてパーセンタイルを設定すると、統計評価に使用する十分なデータがない場合の処理を指定することができます。そのまま統計を評価し、任意でアラームの状態を変更するように指定することもできます。また、サンプルサイズが小さい場合は、メトリクスを無視し、統計的に十分なデータが揃うまで評価を保留することもできます。

パーセンタイルが 0.5 以上 1.00 未満の場合、この設定は、評価期間中にデータポイントが 10/(1-パーセンタイル) を下回ると使用されます。たとえば、この設定は、p99 パーセンタイルのアラームのサンプル数が 1000 を下回ると使用されます。パーセンタイルが 0 以上 0.5 未満の場合、この設定は、データポイントが 10/パーセンタイルを下回ると使用されます。

CloudWatch アラームの一般的な機能

次の機能は、すべての CloudWatch アラームに適用されます。

- 作成できるアラームの数に制限はありません。アラームを作成または更新するには、CloudWatch コンソール、PutMetricAlarm API アクション、または AWS CLI の put-metric-alarm コマンドを使用します。
- アラーム名には UTF-8 文字のみを使用する必要があり、ASCII 制御文字は使用できません。
- 現在設定しているアラームのいずれかまたはすべてを一覧表示したり、特定の状態にあるアラームを一覧表示したりするには、CloudWatch コンソール、DescribeAlarms API アクション、または AWS CLI の describe-alarms コマンドを使用します。
- アラームを無効または有効にするには、DisableAlarmActions および EnableAlarmActions API アクション、または AWS CLI の disable-alarm-actions および enable-alarm-actions コマンドを使用します。
- アラームの動作をテストするには、SetAlarmState API アクションまたは AWS CLI の set-alarm-state コマンドを使用して、アラームを任意の状態に設定します。この一時的な状態変更が持続するのは、次のアラーム比較が行われるときまでです。
- カスタムメトリクスを作成する前に、カスタムメトリクスのアラームを作成できます。アラームを有効にするには、メトリクス名前空間およびメトリクス名に加えて、カスタムメトリクスのディメンションをすべてアラームの定義に含める必要があります。これを行うには、PutMetricAlarm API アクションを使用するか、AWS CLI で put-metric-alarm コマンドを使用します。
- アラームの履歴を表示するには、CloudWatch コンソール、DescribeAlarmHistory API アクション、または AWS CLI の describe-alarm-history コマンドを使用します。CloudWatch は、アラーム

履歴を 30 日間保存します。状態変化ごとに固有のタイムスタンプが付きます。まれに、1 つの状態変化に対して複数の通知が履歴に残されることがあります。タイムスタンプによって状態変化を区別できます。

- [Favorites and recents] (お気に入りと最近のアクセス) オプションでお気に入りのアラームにカーソルを合わせ、その横にある星記号をクリックして、CloudWatch コンソールのナビゲーションペインにお気に入りとして登録できます。
- アラームの評価期間の数を各評価期間の長さで乗算した値が 1 日を超えることはできません。

> **Note**
>
> 特定の状況では、AWS リソースが CloudWatch にメトリクスデータを送信しないことがあります。
> 例えば、Amazon EBS は、Amazon EC2 インスタンスに追加されていない利用可能なボリュームに関するメトリクスデータを送信しないことがあります。これは、そのボリュームでモニターリングするメトリクスの動作がないためです。このようなメトリクスにアラームを設定すると、状態が `INSUFFICIENT_DATA` に変わる場合があります。これは、リソースが動作していないことを示すもので、必ずしも問題があることを意味しているわけではありません。各アラームが欠落データを処理する方法を指定できます。詳細については、「CloudWatch アラームの欠落データの処理の設定」を参照してください。

AWS のサービスに関するベストプラクティスアラームの推奨事項

CloudWatch は、すぐに使えるアラームの推奨事項を提供します。これらは CloudWatch アラームで、他の AWS のサービスによって公開されるメトリクス用に作成することをお勧めするものです。これらの推奨事項は、モニタリングのベストプラクティスに従うためにアラームを設定すべきメトリクスを特定するのに役立ちます。推奨事項では、設定すべきアラームのしきい値も提案されています。これらの推奨事項に従うことで、AWS インフラストラクチャの重要なモニタリングを見逃さないようにできます。

アラームの推奨事項を見つけるには、CloudWatch コンソールの [メトリクス] セクションを使用し、[アラームの推奨事項] フィルタートグルを選択します。コンソールの [推奨されたアラーム] に移動してから推奨アラームを作成すると、CloudWatch はアラーム設定の一部を事前に入力できます。一部の推奨アラームでは、アラームのしきい値も事前に入力されています。また、コンソールを使用して推奨アラームの IaC (Infrastructure-as-Code) アラーム定義をダウンロードし、このコードを使用して、AWS CloudFormation、AWS CLI、または Terraform でアラームを作成することもできます。

推奨アラームの一覧は、推奨アラーム でも確認できます。

作成したアラームには、CloudWatch で作成した他のアラームと同じレートで課金されます。推奨事項の使用には、追加課金は発生しません。詳細については、「Amazon CloudWatch 料金表」をご覧ください。

推奨アラームを見つけて作成する

次の手順に従って、CloudWatch がアラームを設定するように推奨するメトリクスを見つけ、必要に応じてこれらのアラームの 1 つを作成します。最初の手順では、推奨アラームのあるメトリクスを見つける方法と、これらのアラームの 1 つを作成する方法について説明します。

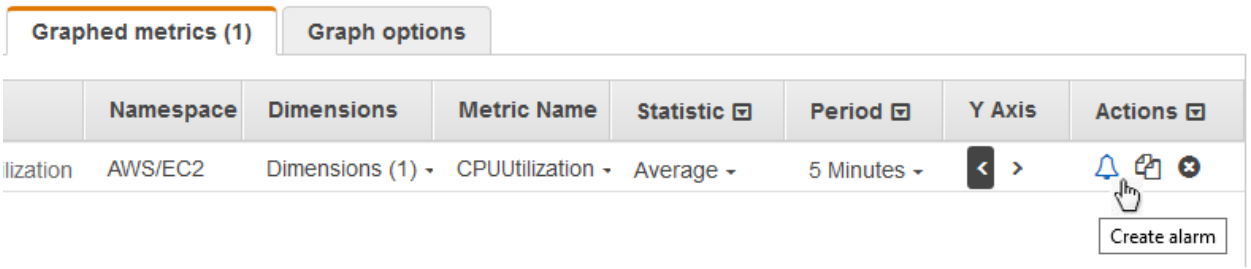
また、AWS/Lambda や AWS/S3 など AWS 名前空間内の推奨アラームのすべてについて IaC (Infrastructure-as-Code) アラーム定義を一括ダウンロードすることもできます。これらの手順は、このトピックで後ほど紹介します。

推奨アラームを含むメトリクスを見つけ、単一の推奨アラームを作成するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで、[Metrics]、[All metrics] を選択します。
3. [メトリクス] テーブルの上にある [アラームの推奨事項] を選択します。

   アラームの推奨事項があり、アカウント内のサービスが公開しているメトリクスのみを含むように、メトリクス名前空間のリストはフィルタリングされます。
4. サービスの名前空間を選択します。

   この名前空間のメトリクスのリストは、アラームの推奨事項があるものだけを含むようにフィルタリングされます。
5. メトリクスのアラームの目的と推奨しきい値を確認するには、[詳細を表示] を選択します。
6. メトリクスのいずれかにアラームを作成するには、以下のいずれかを実行します。
   - コンソールを使用してアラームを作成するには、以下を実行します。
     a. メトリクスのチェックボックスを選択し、[グラフ化したメトリクス] タブを選択します。
     b. アラームアイコンを選択します。



アラームの推奨事項に基づいて、メトリクス名、統計、期間が入力されているアラーム作成ウィザードが表示されます。推奨事項に具体的なしきい値が含まれている場合は、その値も事前に入力されています。

c. [Next] を選択します。

d. [通知] で、アラームが ALARM 状態、OK 状態、または INSUFFICIENT_DATA 状態に移行するときに通知するための SNS トピックを選択します。

同じアラーム状態または複数の異なるアラーム状態について複数の通知を送信するには、[Add notification (通知の追加)] を選択します。

アラームの通知を送信しない場合は、 [削除] を選択します。

e. アラームに伴って Auto Scaling または EC2 アクションを実行するには、該当するボタンを選択し、アラーム状態と実行するアクションを選択します。

f. 完了したら、[次へ] を選択します。

g. アラームの名前と説明を入力します。名前には ASCII 文字のみを使用します。続いて、[次へ] を選択します。

h. [Preview and create (プレビューして作成)] で、情報と条件が正しいことを確認し、[アラームの作成] を選択します。

- IaC (Infrastructure-as-Code) アラーム定義をダウンロードして、AWS CloudFormation、AWS CLI、または Terraform で使用するには、[アラームコードをダウンロード] を選択し、必要な形式を選択します。ダウンロードしたコードには、メトリクス名、統計、およびしきい値の推奨設定が含まれます。

AWS のサービスのすべての推奨アラームの IaC (Infrastructure-as-Code) アラーム定義をダウンロードするには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで、[Metrics]、[All metrics] を選択します。

3. [メトリクス] テーブルの上にある [アラームの推奨事項] を選択します。

   アラームの推奨事項があり、アカウント内のサービスが公開しているメトリクスのみを含むように、メトリクス名前空間のリストはフィルタリングされます。

4. サービスの名前空間を選択します。

   この名前空間のメトリクスのリストは、アラームの推奨事項があるものだけを含むようにフィルタリングされます。

5. [アラームコードをダウンロード] によって、この名前空間のメトリクスに対して推奨されるアラームの数が表示されます。すべての推奨アラームの IaC (Infrastructure-as-Code) アラーム定義をダウンロードするには、[アラームコードのダウンロード] を選択し、必要なコード形式を選択します。

推奨アラーム

以下のセクションでは、ベストプラクティスアラームを設定することをお勧めするメトリクスを一覧表示しています。各メトリクスには、ディメンション、アラームの目的、推奨しきい値、しきい値の根拠、期間の長さとデータポイントの数も表示されます。

一部のメトリクスはリストに 2 回表示されることがあります。これは、そのメトリクスのディメンションの組み合わせによって異なるアラームが推奨される場合に発生します。

アラームを発生させるデータポイント数は、アラームが ALARM 状態になるのに必要な違反データポイントの数です。評価期間数 は、アラームの評価時に考慮される期間の数です。この 2 つの数が同じ場合、期間の値がその数だけ連続してしきい値を超えた場合にのみ、アラームは ALARM 状態になります。アラームを発生させるデータポイント数が評価期間数より少ない場合、そのアラームは「N 件中 M 件」のアラームであり、少なくともアラームを発生させるデータポイント数のデータポイントが、任意の評価期間数のデータポイントセット内で違反していると、アラームは ALARM 状態になります。詳細については、「アラームの評価」を参照してください。

トピック

- Amazon API Gateway
- Amazon EC2 Auto Scaling
- Amazon CloudFront
- Amazon Cognito
- Amazon DynamoDB

- Amazon EBS
- Amazon EC2
- Amazon ElastiCache
- Amazon EC2 (AWS/ElasticGPUs)
- Amazon ECS
- Container Insights を使用する Amazon ECS
- Amazon EFS
- Container Insights を使用する Amazon EKS
- Amazon Kinesis Data Streams
- Lambda
- Lambda Insights
- Amazon VPC (AWS/NATGateway)
- AWS プライベートリンク (AWS/PrivateLinkEndpoints)
- AWS プライベートリンク (AWS/PrivateLinkServices)
- Amazon RDS
- Amazon Route 53 Public Data Plane
- Amazon S3
- S3ObjectLambda
- Amazon SNS
- Amazon SQS
- AWS VPN

Amazon API Gateway

4XXError

ディメンション: ApiName、Stage

アラームの説明: このアラームは、クライアント側エラーの発生率が高いことを検出します。これは、認証パラメータまたはクライアントリクエストパラメータに問題があることを示している可能性があります。また、リソースが削除されたか、存在しないリソースをクライアントがリクエストしている可能性もあります。CloudWatch Logs を有効にして、4XX エラーの原因

となっている可能性のあるエラーがないか確認することを検討してください。さらに、詳細な CloudWatch メトリクスを有効にして、リソースやメソッドごとにこのメトリクスを表示し、エラーの原因を絞り込むことを検討してください。また、設定したスロットリング制限を超えたためにエラーが発生することもあります。レスポンスとログで報告されている 429 エラーの発生率が高く予想外の場合は、このガイドに従ってこの問題をトラブルシューティングしてください。

目的: このアラームは、API Gateway リクエストのクライアント側エラーの発生率が高いことを検出できます。

統計: Average

推奨しきい値: 0.05

しきい値の根拠: 推奨しきい値は、全リクエストの 5% 超が 4XX エラーになっていることを検出します。ただし、リクエストのトラフィックや許容可能なエラー発生率に合わせてしきい値を調整できます。また、履歴データを分析してアプリケーションワークロードの許容エラー発生率を判断し、それに応じてしきい値を調整することもできます。頻繁に発生する 4XX エラーにはアラームが必要です。ただし、しきい値を非常に低く設定すると、アラームの感度が高くなりすぎる可能性があります。

期間: 60

アラームを発生させるデータポイント数: 5

評価期間数: 5

比較演算子: GREATER_THAN_THRESHOLD

5XXError

ディメンション: ApiName、Stage

アラームの説明: このアラームは、サーバー側エラーの発生率が高いことを検出するのに役立ちます。これは、API バックエンド、ネットワーク、または API ゲートウェイとバックエンド API の統合に問題があることを示している可能性があります。このドキュメントは、5xx エラーの原因のトラブルシューティングに役立ちます。

目的: このアラームは、API Gateway リクエストのサーバー側エラーの発生率が高いことを検出できます。

統計: Average

推奨しきい値: 0.05

しきい値の根拠: 推奨しきい値は、全リクエストの 5% 超が 5XX エラーになっていることを検出します。ただし、リクエストのトラフィックや許容可能なエラー発生率に合わせてしきい値を調整できます。また、履歴データを分析してアプリケーションワークロードの許容エラー発生率を判断し、それに応じてしきい値を調整することもできます。頻繁に発生する 5XX エラーにはアラームが必要です。ただし、しきい値を非常に低く設定すると、アラームの感度が高くなりすぎる可能性があります。

期間: 60

アラームを発生させるデータポイント数: 3

評価期間数: 3

比較演算子: GREATER_THAN_THRESHOLD

Count (カウント)

ディメンション: ApiName、Stage

アラームの説明: このアラームは、REST API ステージのトラフィック量が少ないことを検出するのに役立ちます。これは、API を呼び出すアプリケーションに問題があることを示している可能性があります。たとえば、不正なエンドポイントを使用している場合などです。また、API の設定やアクセス許可に問題があり、クライアントがアクセスできないことを示している可能性もあります。

目的: このアラームは、REST API ステージのトラフィック量が予想外に少ないことを検出できます。通常の状態で API が予測可能で一定数のリクエストを受信する場合は、このアラームを作成することをお勧めします。詳細な CloudWatch メトリクスを有効にしていて、メソッドとリソースごとの通常のトラフィック量を予測できる場合は、これに代わるアラームを複数作成して、各リソースとメソッドのトラフィック量の減少をより詳細にモニタリングすることをお勧めします。一定で一貫したトラフィックを想定していない API には、このアラームはお勧めしません。

統計: SampleCount

推奨しきい値: 状況によって異なります。

しきい値の根拠: 履歴データ分析に基づいてしきい値を設定し、API のリクエスト数の予想ベースラインを決定します。しきい値を非常に高い値に設定すると、通常時およびトラフィックが少ないことが予想される時間帯に、アラームの感度が高くなりすぎる可能性があります。逆に、非常

に低い値に設定すると、トラフィック量の異常な減少をアラームが見逃してしまう可能性があります。

期間: 60

アラームを発生させるデータポイント数: 10

評価期間数: 10

比較演算子: LESS_THAN_THRESHOLD

Count (カウント)

ディメンション: ApiName、Stage、Resource、Method

アラームの説明: このアラームは、ステージでの REST API リソースとメソッドのトラフィック量が少ないことを検出するのに役立ちます。これは、API を呼び出すアプリケーションに問題があることを示している可能性があります。例えば、不正なエンドポイントを使用している場合などです。また、API の設定やアクセス許可に問題があり、クライアントがアクセスできないことを示している可能性もあります。

目的: このアラームは、ステージでの REST API リソースとメソッドのトラフィック量が予想外に少ないことを検出できます。通常の状態で API が予測可能で一定数のリクエストを受信する場合は、このアラームを作成することをお勧めします。一定で一貫したトラフィックを想定していない API には、このアラームはお勧めしません。

統計: SampleCount

推奨しきい値: 状況によって異なります。

しきい値の根拠: 履歴データ分析に基づいてしきい値を設定し、API のリクエスト数の予想ベースラインを決定します。しきい値を非常に高い値に設定すると、通常時およびトラフィックが少ないことが予想される時間帯に、アラームの感度が高くなりすぎる可能性があります。逆に、非常に低い値に設定すると、トラフィック量の異常な減少をアラームが見逃してしまう可能性があります。

期間: 60

アラームを発生させるデータポイント数: 10

評価期間数: 10

比較演算子: LESS_THAN_THRESHOLD

Count (カウント)

ディメンション: ApiId、Stage

アラームの説明: このアラームは、HTTP API ステージのトラフィック量が少ないことを検出するのに役立ちます。これは、API を呼び出すアプリケーションに問題があることを示している可能性があります。例えば、不正なエンドポイントを使用している場合などです。また、API の設定やアクセス許可に問題があり、クライアントがアクセスできないことを示している可能性もあります。

目的: このアラームは、HTTP API ステージのトラフィック量が予想外に少ないことを検出できます。通常の状態で API が予測可能で一定数のリクエストを受信する場合は、このアラームを作成することをお勧めします。詳細な CloudWatch メトリクスを有効にしていて、ルートごとの通常のトラフィック量を予測できる場合は、これに代わるアラームを複数作成して、各ルートのトラフィック量の減少をより詳細にモニタリングすることをお勧めします。一定で一貫したトラフィックを想定していない API には、このアラームはお勧めしません。

統計: SampleCount

推奨しきい値: 状況によって異なります。

しきい値の根拠: 履歴データ分析に基づいてしきい値を設定し、API のリクエスト数の予想ベースラインを決定します。しきい値を非常に高い値に設定すると、通常時およびトラフィックが少ないことが予想される時間帯に、アラームの感度が高くなりすぎる可能性があります。逆に、非常に低い値に設定すると、トラフィック量の異常な減少をアラームが見逃してしまう可能性があります。

期間: 60

アラームを発生させるデータポイント数: 10

評価期間数: 10

比較演算子: LESS_THAN_THRESHOLD

Count (カウント)

ディメンション: ApiId、Stage、Resource、Method

アラームの説明: このアラームは、ステージ内の HTTP API ルートのトラフィック量が少ないことを検出するのに役立ちます。これは、API を呼び出すアプリケーションに問題があることを示

している可能性があります。例えば、不正なエンドポイントを使用している場合などです。また、API の設定やアクセス許可に問題があり、クライアントがアクセスできないことを示している可能性もあります。

目的: このアラームは、ステージ内の HTTP API ルートのトラフィック量が予想外に少ないことを検出できます。通常の状態で API が予測可能で一定数のリクエストを受信する場合は、このアラームを作成することをお勧めします。一定で一貫したトラフィックを想定していない API には、このアラームはお勧めしません。

統計: SampleCount

推奨しきい値: 状況によって異なります。

しきい値の根拠: 履歴データ分析に基づいてしきい値を設定し、API のリクエスト数の予想ベースラインを決定します。しきい値を非常に高い値に設定すると、通常時およびトラフィックが少ないことが予想される時間帯に、アラームの感度が高くなりすぎる可能性があります。逆に、非常に低い値に設定すると、トラフィック量の異常な減少をアラームが見逃してしまう可能性があります。

期間: 60

アラームを発生させるデータポイント数: 10

評価期間数: 10

比較演算子: LESS_THAN_THRESHOLD

IntegrationLatency

ディメンション: ApiId、Stage

アラームの説明: このアラームは、あるステージの API リクエストの統合レイテンシーが大きいかどうかを検出するのに役立ちます。`IntegrationLatency` メトリクス値を、Lambda 統合の `Duration` メトリクスなど、バックエンドの対応するレイテンシーメトリクスと関連付けることができます。これにより、パフォーマンスの問題が原因で API バックエンドがクライアントからのリクエストを処理するのに時間がかかっているのか、それとも初期化やコールドスタートによるその他のオーバーヘッドがあるのかを判断できます。さらに、API の CloudWatch Logs を有効にして、レイテンシーが大きい問題を引き起こしている可能性のあるエラーがないかログを確認することを検討してください。さらに、詳細な CloudWatch メトリクスを有効にして、ルートごとにこのメトリクスを表示することを検討してください。これにより、統合レイテンシーの原因を絞り込むことができます。

目的: このアラームは、ステージ内の API Gateway リクエストの統合レイテンシーが大きいことを検出できます。WebSocket API にはこのアラームをお勧めしますが、HTTP API ではオプションと考えています。これは、レイテンシーメトリクスに関する個別のアラーム推奨事項がすでに設定されているためです。詳細な CloudWatch メトリクスを有効にしていて、ルートごとに統合レイテンシーのパフォーマンス要件が異なる場合は、これに代わるアラームを複数作成して、各ルートの統合レイテンシーをより詳細にモニタリングすることをお勧めします。

統計: p90

推奨しきい値: 2000.0

しきい値の根拠: 推奨しきい値は、すべての API ワークロードに適しているわけではありません。ただし、しきい値の初期値として使用が可能です。その後、ワークロードや許容可能なレイテンシー、パフォーマンス、および API に対する SLA 要件に基づいて異なるしきい値を選択できます。一般的に API のレイテンシーが大きいことを許容できる場合は、しきい値を高く設定してアラームの感度を低くします。一方、API がほぼリアルタイムで応答することが期待される場合は、しきい値を低く設定します。また、履歴データを分析してアプリケーションワークロードのレイテンシーの予想ベースラインを判断し、それに応じてしきい値を調整することもできます。

期間: 60

アラームを発生させるデータポイント数: 5

評価期間数: 5

比較演算子: GREATER_THAN_OR_EQUAL_TO_THRESHOLD

IntegrationLatency

ディメンション: ApiId、Stage、Route

アラームの説明: このアラームは、ステージ内のルートに対する WebSocket API リクエストの統合レイテンシーが大きいかどうかを検出するのに役立ちます。`IntegrationLatency` メトリクス値を、Lambda 統合の `Duration` メトリクスなど、バックエンドの対応するレイテンシーメトリクスと関連付けることができます。これにより、パフォーマンスの問題が原因で API バックエンドがクライアントからのリクエストを処理するのに時間がかかっているのか、それとも初期化やコールドスタートによるその他のオーバーヘッドがあるのかを判断できます。さらに、API の CloudWatch Logs を有効にして、レイテンシーが大きい問題を引き起こしている可能性のあるエラーがないかログを確認することを検討してください。

目的:このアラームは、ステージ内のルートに対する API Gateway リクエストの統合レイテンシーが大きいことを検出できます。

統計: p90

推奨しきい値: 2000.0

しきい値の根拠: 推奨しきい値は、すべての API ワークロードに適しているわけではありません。ただし、しきい値の初期値として使用が可能です。その後、ワークロードや許容可能なレイテンシー、パフォーマンス、および API に対する SLA 要件に基づいて異なるしきい値を選択できます。一般的に API のレイテンシーが大きいことを許容できる場合は、しきい値を高く設定してアラームの感度を低くできます。一方、API がほぼリアルタイムで応答することが期待される場合は、しきい値を低く設定します。また、履歴データを分析してアプリケーションワークロードのレイテンシーの予想ベースラインを判断し、それに応じてしきい値を調整することもできます。

期間: 60

アラームを発生させるデータポイント数: 5

評価期間数: 5

比較演算子: GREATER_THAN_OR_EQUAL_TO_THRESHOLD

レイテンシー

ディメンション: ApiName、Stage

アラームの説明: このアラームは、ステージ内のレイテンシーが大きいことを検出します。`IntegrationLatency` メトリクス値を調べて API バックエンドのレイテンシーをチェックします。2 つのメトリクスがほぼ一致している場合、API バックエンドがレイテンシー増加の原因となるため、問題がないか調査する必要があります。CloudWatch Logs を有効にして、レイテンシーが大きいことの原因となっている可能性のあるエラーをチェックすることも検討してください。さらに、詳細な CloudWatch メトリクスを有効にして、リソースとメソッドごとにこのメトリクスを表示して、レイテンシーの原因を絞り込むことを検討してください。該当する場合は、「Lambda でのトラブルシューティング」または「エッジ最適化 API エンドポイントのトラブルシューティング」ガイドを参照してください。

目的: このアラームは、ステージ内の API Gateway リクエストのレイテンシーが大きいことを検出できます。詳細な CloudWatch メトリクスを有効にしていて、メソッドとリソースごとにレイ

テンシーのパフォーマンス要件が異なる場合は、これに代わるアラームを複数作成して、各リソースとメソッドのレイテンシーをより詳細にモニタリングすることをお勧めします。

統計: p90

推奨しきい値: 2500.0

しきい値の根拠: 推奨しきい値は、すべての API ワークロードに適しているわけではありません。ただし、しきい値の初期値として使用が可能です。その後、ワークロードや許容可能なレイテンシー、パフォーマンス、および API に対する SLA 要件に基づいて異なるしきい値を選択できます。一般的に API のレイテンシーが大きいことを許容できる場合は、しきい値を高く設定してアラームの感度を低くできます。一方、API がほぼリアルタイムで応答することが期待される場合は、しきい値を低く設定します。また、履歴データを分析してアプリケーションワークロードの予想ベースラインレイテンシーを判断し、それに応じてしきい値を調整することもできます。

期間: 60

アラームを発生させるデータポイント数: 5

評価期間数: 5

比較演算子: GREATER_THAN_OR_EQUAL_TO_THRESHOLD

レイテンシー

ディメンション: ApiName、Stage、Resource、Method

アラームの説明: このアラームは、ステージ内のリソースとメソッドのレイテンシーが大きいことを検出します。`IntegrationLatency` メトリクス値を調べて API バックエンドのレイテンシーをチェックします。2 つのメトリクスがほぼ一致している場合、API バックエンドがレイテンシー増加の原因となるため、パフォーマンスの問題がないか調査する必要があります。CloudWatch Logs を有効にして、レイテンシーが大きいことの原因となっている可能性のあるエラーがないかをチェックすることも検討してください。該当する場合は、「Lambda でのトラブルシューティング」または「エッジ最適化 API エンドポイントのトラブルシューティング」ガイドを参照することもできます。

目的: このアラームは、ステージ内のリソースとメソッドに対する API Gateway リクエストの統合レイテンシーが大きいことを検出できます。

統計: p90

推奨しきい値: 2500.0

しきい値の根拠: 推奨しきい値は、すべての API ワークロードに適しているわけではありません。ただし、しきい値の初期値として使用が可能です。その後、ワークロードや許容可能なレイテンシー、パフォーマンス、および API に対する SLA 要件に基づいて異なるしきい値を選択できます。一般的に API のレイテンシーが大きいことを許容できる場合は、しきい値を高く設定してアラームの感度を低くできます。一方、API がほぼリアルタイムで応答することが期待される場合は、しきい値を低く設定します。また、履歴データを分析してアプリケーションワークロードのレイテンシーの予想ベースラインを判断し、それに応じてしきい値を調整することもできます。

期間: 60

アラームを発生させるデータポイント数: 5

評価期間数: 5

比較演算子: GREATER_THAN_OR_EQUAL_TO_THRESHOLD

レイテンシー

ディメンション: ApiId、Stage

アラームの説明: このアラームは、ステージ内のレイテンシーが大きいことを検出します。`IntegrationLatency` メトリクス値を調べて API バックエンドのレイテンシーをチェックします。2 つのメトリクスがほぼ一致している場合、API バックエンドがレイテンシー増加の原因となるため、パフォーマンスの問題がないか調査する必要があります。CloudWatch Logs を有効にして、レイテンシーが大きいことの原因となっている可能性のあるエラーがないかをチェックすることも検討してください。さらに、詳細な CloudWatch メトリクスを有効にして、ルートごとにこのメトリクスを表示して、レイテンシーの原因を絞り込むことを検討してください。該当する場合は、「Lambda 統合でのトラブルシューティング」ガイドも参照できます。

目的: このアラームは、ステージ内の API Gateway リクエストのレイテンシーが大きいことを検出できます。詳細な CloudWatch メトリクスを有効にしていて、ルートごとにレイテンシーのパフォーマンス要件が異なる場合は、これに代わるアラームを複数作成して、各ルートのレイテンシーをより詳細にモニタリングすることをお勧めします。

統計: p90

推奨しきい値: 2500.0

しきい値の根拠: 推奨しきい値は、すべての API ワークロードに適しているわけではありません。ただし、しきい値の初期値として使用が可能です。その後、ワークロードや許容可能なレイ

テンシー、パフォーマンス、および API に対する SLA 要件に基づいて異なるしきい値を選択できます。一般的に API のレイテンシーが大きいことを許容できる場合は、しきい値を高く設定して感度を低くできます。一方、API がほぼリアルタイムで応答することが期待される場合は、しきい値を低く設定します。また、履歴データを分析してアプリケーションワークロードのレイテンシーの予想ベースラインを判断し、それに応じてしきい値を調整することもできます。

期間: 60

アラームを発生させるデータポイント数: 5

評価期間数: 5

比較演算子: GREATER_THAN_OR_EQUAL_TO_THRESHOLD

レイテンシー

ディメンション: ApiId、Stage、Resource、Method

アラームの説明: このアラームは、ステージ内のルートのレイテンシーが大きいことを検出します。`IntegrationLatency` メトリクス値を調べて API バックエンドのレイテンシーをチェックします。2 つのメトリクスがほぼ一致している場合、API バックエンドがレイテンシー増加の原因となるため、パフォーマンスの問題がないか調査する必要があります。CloudWatch Logs を有効にして、レイテンシーが大きいことの原因となっている可能性のあるエラーがないかをチェックすることも検討してください。該当する場合は、「Lambda 統合でのトラブルシューティング」ガイドも参照できます。

目的:このアラームは、ステージ内のルートに対する API Gateway リクエストのレイテンシーが大きいことを検出するために使用されます。

統計: p90

推奨しきい値: 2500.0

しきい値の根拠: 推奨しきい値は、すべての API ワークロードに適しているわけではありません。ただし、しきい値の初期値として使用が可能です。その後、ワークロードや許容可能なレイテンシー、パフォーマンス、および API に対する SLA 要件に基づいて異なるしきい値を選択できます。一般的に API のレイテンシーが大きいことを許容できる場合は、しきい値を高く設定してアラームの感度を低くできます。一方、API がほぼリアルタイムで応答することが期待される場合は、しきい値を低く設定します。また、履歴データを分析してアプリケーションワークロードのレイテンシーの予想ベースラインを判断し、それに応じてしきい値を調整することもできます。

期間: 60

アラームを発生させるデータポイント数: 5

評価期間数: 5

比較演算子: GREATER_THAN_OR_EQUAL_TO_THRESHOLD

4xx

ディメンション: ApiId、Stage

アラームの説明: このアラームは、クライアント側エラーの発生率が高いことを検出します。これは、認証パラメータまたはクライアントリクエストパラメータに問題があることを示している可能性があります。また、ルートが削除されたか、API に存在しないルートをクライアントがリクエストしている可能性もあります。CloudWatch Logs を有効にして、4xx エラーの原因となっている可能性のあるエラーがないか確認することを検討してください。さらに、詳細な CloudWatch メトリクスを有効にして、ルートごとにこのメトリクスを表示して、エラーの原因の絞り込みに役立てることを検討してください。また、設定したスロットリング制限を超えたためにエラーが発生することもあります。レスポンスとログで報告されている 429 エラーの発生率が高く予想外の場合は、このガイドに従ってこの問題をトラブルシューティングしてください。

目的: このアラームは、API Gateway リクエストのクライアント側エラーの発生率が高いことを検出できます。

統計: Average

推奨しきい値: 0.05

しきい値の根拠: 推奨しきい値は、全リクエストの 5% 超が 4xx エラーになっていることを検出します。ただし、リクエストのトラフィックや許容可能なエラー発生率に合わせてしきい値を調整できます。また、履歴データを分析してアプリケーションワークロードの許容エラー発生率を判断し、それに応じてしきい値を調整することもできます。頻繁に発生する 4xx エラーにはアラームが必要です。ただし、しきい値を非常に低く設定すると、アラームの感度が高くなりすぎる可能性があります。

期間: 60

アラームを発生させるデータポイント数: 5

評価期間数: 5

比較演算子: GREATER_THAN_THRESHOLD

5xx

ディメンション: ApiId、Stage

アラームの説明: このアラームは、サーバー側エラーの発生率が高いことを検出するのに役立ちます。これは、API バックエンド、ネットワーク、または API ゲートウェイとバックエンド API の統合に問題があることを示している可能性があります。このドキュメントは、5xx エラーの原因のトラブルシューティングに役立ちます。

目的: このアラームは、API Gateway リクエストのサーバー側エラーの発生率が高いことを検出できます。

統計: Average

推奨しきい値: 0.05

しきい値の根拠: 推奨しきい値は、全リクエストの 5% 超が 5xx エラーになっていることを検出します。ただし、リクエストのトラフィックや許容可能なエラー発生率に合わせてしきい値を調整できます。また、履歴データを分析してアプリケーションワークロードの許容エラー発生率を判断し、それに応じてしきい値を調整することもできます。頻繁に発生する 5xx エラーにはアラームが必要です。ただし、しきい値を非常に低く設定すると、アラームの感度が高くなりすぎる可能性があります。

期間: 60

アラームを発生させるデータポイント数: 3

評価期間数: 3

比較演算子: GREATER_THAN_THRESHOLD

MessageCount

ディメンション: ApiId、Stage

アラームの説明: このアラームは、WebSocket API ステージのトラフィック量が少ないことを検出するのに役立ちます。これは、クライアントが API を呼び出す際に不正なエンドポイントを使用するなどの問題や、バックエンドがクライアントにメッセージを送信する際の問題を示している可能性があります。また、API の設定やアクセス許可に問題があり、クライアントがアクセスできないことを示している可能性もあります。

目的: このアラームは、WebSocket API ステージのトラフィック量が予想外に少ないことを検出できます。通常の状態で API が予測可能で一定数のメッセージを送受信する場合は、このアラームを作成することをお勧めします。詳細な CloudWatch メトリクスを有効にしていて、ルートごとの通常のトラフィック量を予測できる場合は、これに代わるアラームを複数作成して、各ルートのトラフィック量の減少をより詳細にモニタリングすることをお勧めします。一定で一貫したトラフィックを想定していない API には、このアラームはお勧めしません。

統計: SampleCount

推奨しきい値: 状況によって異なります。

しきい値の根拠: 履歴データ分析に基づいてしきい値を設定し、API のメッセージ数の予想ベースラインを決定します。しきい値を非常に高い値に設定すると、通常時およびトラフィックが少ないことが予想される時間帯に、アラームの感度が高くなりすぎる可能性があります。逆に、非常に低い値に設定すると、トラフィック量の異常な減少をアラームが見逃してしまう可能性があります。

期間: 60

アラームを発生させるデータポイント数: 10

評価期間数: 10

比較演算子: LESS_THAN_THRESHOLD

MessageCount

ディメンション: ApiId、Stage、Route

アラームの説明: このアラームは、ステージ内の WebSocket API ルートのトラフィック量が少ないことを検出するのに役立ちます。これは、クライアントが API を呼び出す際に不正なエンドポイントを使用するなどの問題や、バックエンドがクライアントにメッセージを送信する際の問題を示している可能性があります。また、API の設定やアクセス許可に問題があり、クライアントがアクセスできないことを示している可能性もあります。

目的: このアラームは、ステージ内の WebSocket API ルートのトラフィック量が予想外に少ないことを検出できます。通常の状態で API が予測可能で一定数のメッセージを送受信する場合は、このアラームを作成することをお勧めします。一定で一貫したトラフィックを想定していない API には、このアラームはお勧めしません。

統計: SampleCount

推奨しきい値: 状況によって異なります。

しきい値の根拠: 履歴データ分析に基づいてしきい値を設定し、API のメッセージ数の予想ベースラインを決定します。しきい値を非常に高い値に設定すると、通常時およびトラフィックが少ないことが予想される時間帯に、アラームの感度が高くなりすぎる可能性があります。逆に、非常に低い値に設定すると、トラフィック量の異常な減少をアラームが見逃してしまう可能性があります。

期間: 60

アラームを発生させるデータポイント数: 10

評価期間数: 10

比較演算子: LESS_THAN_THRESHOLD

ClientError

ディメンション: ApiId、Stage

アラームの説明: このアラームは、クライアントエラーの発生率が高いことを検出します。これは、認証パラメータまたはメッセージパラメータに問題があることを示している可能性があります。また、ルートが削除されたか、API に存在しないルートをクライアントがリクエストしている可能性もあります。CloudWatch Logs を有効にして、4xx エラーの原因となっている可能性のあるエラーがないか確認することを検討してください。さらに、詳細な CloudWatch メトリクスを有効にして、ルートごとにこのメトリクスを表示して、エラーの原因の絞り込みに役立てることを検討してください。また、設定したスロットリング制限を超えたためにエラーが発生することもあります。レスポンスとログで報告されている 429 エラーの発生率が高く予想外の場合は、このガイドに従ってこの問題をトラブルシューティングしてください。

目的: このアラームは、WebSocket API Gateway メッセージのクライアントエラーの発生率が高いことを検出できます。

統計: Average

推奨しきい値: 0.05

しきい値の根拠: 推奨しきい値は、全リクエストの 5% 超が 4xx エラーになっていることを検出します。リクエストのトラフィックや許容可能なエラー発生率に合わせてしきい値を調整できます。また、履歴データを分析してアプリケーションワークロードの許容エラー発生率を判断し、それに応じてしきい値を調整することもできます。頻繁に発生する 4xx エラーにはアラームが必

要です。ただし、しきい値を非常に低く設定すると、アラームの感度が高くなりすぎる可能性があります。

期間: 60

アラームを発生させるデータポイント数: 5

評価期間数: 5

比較演算子: GREATER_THAN_THRESHOLD

ExecutionError

ディメンション: ApiId、Stage

アラームの説明:このアラームは、実行エラーの発生率が高いことを検出するのに役立ちます。これは、統合、アクセス許可の問題、または統合の呼び出しを正常に実行できないその他の要因 (統合がスロットリングや削除されるなど) による 5xx エラーが原因である可能性があります。API の CloudWatch Logs を有効にし、ログでエラーの種類と原因を確認することを検討してください。さらに、詳細な CloudWatch メトリクスを有効にして、ルートごとにこのメトリクスを表示して、エラーの原因の絞り込みに役立てることを検討してください。このドキュメントも、接続エラーの原因のトラブルシューティングに役立ちます。

目的: このアラームは、WebSocket API Gateway メッセージの実行エラーの発生率が高いことを検出できます。

統計: Average

推奨しきい値: 0.05

しきい値の根拠: 推奨しきい値は、全リクエストの 5% 超が実行エラーになっていることを検出します。リクエストのトラフィックや許容可能なエラー発生率に合わせてしきい値を調整できます。履歴データを分析してアプリケーションワークロードの許容エラー発生率を判断し、それに応じてしきい値を調整できます。頻繁に発生する実行エラーにはアラームが必要です。ただし、しきい値を非常に低く設定すると、アラームの感度が高くなりすぎる可能性があります。

期間: 60

アラームを発生させるデータポイント数: 3

評価期間数: 3

比較演算子: GREATER_THAN_THRESHOLD

Amazon EC2 Auto Scaling

GroupInServiceCapacity

ディメンション: AutoScalingGroupName

アラームの説明: このアラームは、グループ内の容量がワークロードに必要な希望容量を下回ったことを検出するのに役立ちます。トラブルシューティングを行うには、スケーリングアクティビティの起動に障害がないかどうかをチェックし、必要な容量設定が正しいことを確認します。

目的: このアラームは、起動の失敗または中断が原因で 自動スケーリンググループの可用性が低下していることを検出できます。

統計: Average

推奨しきい値: 状況によって異なります。

しきい値の根拠: しきい値は、ワークロードを実行するのに必要な最小容量にするべきです。ほとんどの場合、GroupDesiredCapacity メトリクスと一致するように設定できます。

期間: 60

アラームを発生させるデータポイント数: 10

評価期間数: 10

比較演算子: LESS_THAN_THRESHOLD

Amazon CloudFront

5xxErrorRate

ディメンション: DistributionId、Region=Global

アラームの説明: このアラームは、オリジンサーバーからの 5xx エラーレスポンスの割合をモニタリングし、CloudFront サービスに問題が発生しているかどうかを検出するのに役立ちます。サーバーの問題点を理解するのに役立つ情報については、「オリジンからのエラーレスポンスのトラブルシューティング」を参照してください。また、詳細なエラーメトリクスを取得するには、追加のメトリクスをオンにしてください。

目的: このアラームは、オリジンサーバーからのリクエストの処理に関する問題、または CloudFront とオリジンサーバー間の通信の問題を検出するために使用されます。

統計: Average

推奨しきい値: 状況によって異なります。

しきい値の根拠: このアラームの推奨しきい値は、5xx レスポンスの許容度によって大きく異なります。履歴データと傾向を分析し、それに応じてしきい値を設定できます。5xx エラーは一時的な問題が原因で発生する可能性があるため、アラームの感度が高すぎないように、しきい値を 0 より大きい値に設定することをお勧めします。

期間: 60

アラームを発生させるデータポイント数: 5

評価期間数: 5

比較演算子: GREATER_THAN_THRESHOLD

OriginLatency

ディメンション: DistributionId、Region=Global

アラームの説明: このアラームは、オリジンサーバーの応答に時間がかかりすぎていないかをモニタリングするのに役立ちます。サーバーの応答に時間がかかりすぎると、タイムアウトになる可能性があります。`OriginLatency` が一貫して高い場合は、「オリジンサーバーでアプリケーションからの遅延したレスポンスを見つけて修正する」を参照してください。

目的: このアラームは、オリジンサーバーが応答に時間がかかりすぎるという問題を検出するために使用されます。

統計: p90

推奨しきい値: 状況によって異なります。

しきい値の根拠: オリジンレスポンスタイムアウトの約 80% の値を計算し、その結果をしきい値として使用してください。このメトリクスが常にオリジンレスポンスタイムアウト値に近い場合、504 エラーが発生し始める可能性があります。

期間: 60

アラームを発生させるデータポイント数: 5

評価期間数: 5

比較演算子: GREATER_THAN_THRESHOLD

FunctionValidationErrors

ディメンション: DistributionId、FunctionName、Region=Global

アラームの説明: このアラームは、CloudFront 関数からの検証エラーをモニタリングし、解決するための措置を講じるのに役立ちます。CloudWatch 関数ログを分析し、関数コードを調べて問題の根本原因を見つけて解決します。CloudFront Functions のよくある設定ミスを理解するには、「エッジ関数に対する制限」を参照してください。

目的: このアラームは、CloudFront 関数からの検証エラーを検出するために使用されます。

統計: Sum

推奨しきい値: 0.0

しきい値の根拠: 0 より大きい値は検証エラーを示します。検証エラーがあると CloudFront 関数が CloudFront に引き渡されるときに問題が発生していることになるため、しきい値を 0 に設定することをお勧めします。例えば、CloudFront はリクエストを処理するために HTTP ホストヘッダーを必要とします。ユーザーは CloudFront 関数コード内のホストヘッダーを削除できてしまいます。しかし、CloudFront がレスポンスを受け取り、ホストヘッダーが見つからない場合、CloudFront は検証エラーをスローします。

期間: 60

アラームを発生させるデータポイント数: 2

評価期間数: 2

比較演算子: GREATER_THAN_THRESHOLD

FunctionExecutionErrors

ディメンション: DistributionId、FunctionName、Region=Global

アラームの説明: このアラームは、CloudFront 関数からの実行エラーをモニタリングし、解決するための措置を講じるのに役立ちます。CloudWatch 関数ログを分析し、関数コードを調べて問題の根本原因を見つけて解決します。

目的: このアラームは、CloudFront 関数からの実行エラーを検出するために使用されます。

統計: Sum

推奨しきい値: 0.0

しきい値の根拠: 実行エラーは実行時に発生するコードに問題があることを示しているため、しきい値を 0 に設定することをお勧めします。

期間: 60

アラームを発生させるデータポイント数: 5

評価期間数: 5

比較演算子: GREATER_THAN_THRESHOLD

FunctionThrottles

ディメンション: DistributionId、FunctionName、Region=Global

アラームの説明: このアラームは、CloudFront 関数がスロットリングされているかどうかをモニタリングするのに役立ちます。関数がスロットリングされている場合は、実行に時間がかかりすぎていることを意味します。関数のスロットリングを避けるには、関数コードの最適化を検討してください。

目的: このアラームは、スムーズなカスタマーエクスペリエンスのために、CloudFront 関数がスロットリングされていることを検出し、問題に対応して解決できます。

統計: Sum

推奨しきい値: 0.0

しきい値の根拠: 関数のスロットリングをより早く解決できるように、しきい値を 0 に設定することをおすすめします。

期間: 60

アラームを発生させるデータポイント数: 5

評価期間数: 5

比較演算子: GREATER_THAN_THRESHOLD

Amazon Cognito

SignUpThrottles

ディメンション: UserPool、UserPoolClient

アラームの説明: このアラームは、スロットリングされたリクエストの数をモニタリングします。ユーザーが定常的にスロットリングされている場合は、サービスクォータの引き上げをリクエストして制限を引き上げる必要があります。クォータの引き上げをリクエストする方法については、「Amazon Cognito のクォータ」を参照してください。事前にアクションを実行するには、クォータの使用状況を追跡することを検討してください。

目的: このアラームは、サインアップリクエストのスロットリングの発生をモニタリングするのに役立ちます。これにより、サインアップでのネガティブな体験の増加を抑えるためのアクションをいつ取るべきかがわかります。リクエストのスロットリングが続くことは、ユーザーがサインアップでネガティブな体験をするということです。

統計: Sum

推奨しきい値: 状況によって異なります。

しきい値の根拠: 適切にプロビジョニングされたユーザープールでは、複数のデータポイントにまたがるスロットリングは発生しないはずです。そのため、予想されるワークロードの通常のしきい値はゼロでなければなりません。バーストが頻繁に発生する不規則なワークロードの場合は、履歴データを分析してアプリケーションのワークロードの許容可能なスロットリングを決定し、それに応じてしきい値を調整できます。スロットリングされたリクエストは、アプリケーションへの影響を最小限に抑えるために再試行する必要があります。

期間: 60

アラームを発生させるデータポイント数: 5

評価期間数: 5

比較演算子: GREATER_THAN_THRESHOLD

SignInThrottles

ディメンション: UserPool、UserPoolClient

アラームの説明: このアラームは、スロットリングされたユーザー認証リクエストの数をモニタリングします。ユーザーが定常的にスロットリングされている場合は、サービスクォータの引き上げをリクエストして制限を引き上げることが必要な場合があります。クォータの引き上げをリクエストする方法については、「Amazon Cognito のクォータ」を参照してください。事前にアクションを実行するには、クォータの使用状況を追跡することを検討してください。

目的: このアラームは、サインインリクエストのスロットリングの発生をモニタリングするのに役立ちます。これにより、サインインでのネガティブな体験の増加を抑えるためのアクションを

いつ取るべきかがわかります。リクエストのスロットリングが続くことは、ユーザー認証で嫌な体験をするということです。

統計: Sum

推奨しきい値: 状況によって異なります。

しきい値の根拠: 適切にプロビジョニングされたユーザープールでは、複数のデータポイントにまたがるスロットリングは発生しないはずです。そのため、予想されるワークロードの通常のしきい値はゼロでなければなりません。バーストが頻繁に発生する不規則なワークロードの場合は、履歴データを分析してアプリケーションのワークロードの許容可能なスロットリングを決定し、それに応じてしきい値を調整できます。スロットリングされたリクエストは、アプリケーションへの影響を最小限に抑えるために再試行する必要があります。

期間: 60

アラームを発生させるデータポイント数: 5

評価期間数: 5

比較演算子: GREATER_THAN_THRESHOLD

TokenRefreshThrottles

ディメンション: UserPool、UserPoolClient

アラームの説明: しきい値は、リクエストのトラフィックに合わせて設定できるほか、トークン更新リクエストの許容可能なスロットリングに合わせて設定することもできます。スロットリングは、リクエストが多すぎる場合にシステムを保護するために使用されます。ただし、通常のトラフィックに対するプロビジョニングが不足していないかどうかもモニタリングすることが重要です。履歴データを分析してアプリケーションのワークロードの許容可能なスロットリングを確認し、アラームのしきい値が許容可能なスロットリングレベルよりも高くなるように調整できます。スロットリングされたリクエストは一時的なものなので、アプリケーション/サービス側で再試行する必要があります。そのため、しきい値が非常に低いと、アラームが敏感になる可能性があります。

目的: このアラームは、トークン更新リクエストのスロットリングの発生をモニタリングするのに役立ちます。これにより、問題が発生する可能性を軽減するためのアクションをいつ取るべきかがわかり、スムーズなユーザーエクスペリエンスと認証システムの健全性と信頼性を確保できます。リクエストのスロットリングが続くことは、ユーザー認証で嫌な体験をするということです。

統計: Sum

推奨しきい値: 状況によって異なります。

しきい値の根拠: しきい値は、リクエストのトラフィックやトークン更新リクエストの許容可能なスロットリングに合わせて設定/調整することもできます。スロットリングは、リクエストが多すぎる場合にシステムを保護するためのものですが、通常のトラフィックに対するプロビジョニングが不足していないかをモニタリングし、影響の原因となっているかどうかを確認することが重要です。履歴データを分析してアプリケーションのワークロードの許容可能なスロットリングを確認し、しきい値が許容可能なスロットリングレベルよりも高くなるように調整することもできます。スロットリングされたリクエストは一時的なものなので、アプリケーション/サービス側で再試行する必要があります。そのため、しきい値が非常に低いと、アラームが敏感になる可能性があります。

期間: 60

アラームを発生させるデータポイント数: 5

評価期間数: 5

比較演算子: GREATER_THAN_THRESHOLD

FederationThrottles

ディメンション: UserPool、UserPoolClient、IdentityProvider

アラームの説明: このアラームは、スロットリングされた ID フェデレーションリクエストの数をモニタリングします。定常的にスロットリングがある場合は、サービスクォータの引き上げをリクエストして制限を引き上げる必要があることを示している可能性があります。クォータの引き上げをリクエストする方法については、「Amazon Cognito のクォータ」を参照してください。

目的: このアラームは、ID フェデレーションリクエストのスロットリングの発生をモニタリングするのに役立ちます。これにより、パフォーマンスのボトルネックや設定ミスに事前に対応し、ユーザーに認証がスムーズだと感じてもらうことができます。リクエストのスロットリングが続くことは、ユーザー認証で嫌な体験をするということです。

統計: Sum

推奨しきい値: 状況によって異なります。

しきい値の根拠: しきい値は、リクエストのトラフィックに合わせて設定できるほか、ID フェデレーションリクエストの許容可能なスロットリングに合わせて設定することもできます。スロッ

トリングは、リクエストが多すぎる場合にシステムを保護するために使用されます。ただし、通常のトラフィックに対するプロビジョニングが不足していないかどうかもモニタリングすることが重要です。履歴データを分析してアプリケーションワークロードの許容可能なスロットリングを確認し、しきい値を許容可能なスロットリングレベルを超える値に設定できます。スロットリングされたリクエストは一時的なものなので、アプリケーション/サービス側で再試行する必要があります。そのため、しきい値が非常に低いと、アラームが敏感になる可能性があります。

期間: 60

アラームを発生させるデータポイント数: 5

評価期間数: 5

比較演算子: GREATER_THAN_THRESHOLD

Amazon DynamoDB

AccountProvisionedReadCapacityUtilization

ディメンション: なし

アラームの説明: このアラームは、アカウントの読み取り容量がプロビジョニングされた制限に達したかどうかを検出します。このような場合は、読み取り容量使用量のアカウントクォータを引き上げることができます。Service Quotas を使用して、読み取り容量ユニットの現在のクォータを表示し、引き上げをリクエストできます。

目的: このアラームは、アカウントの読み込み容量使用率がプロビジョニングされた読み取り容量使用率に近づいているかどうかを検出できます。使用率が上限に達すると、DynamoDB は読み取りリクエストのスロットリングを開始します。

統計: Maximum

推奨しきい値: 80.0

しきい値の根拠: スロットリングを回避するために、しきい値を 80% に設定して、容量が満杯になる前にアクション (アカウント制限の引き上げなど) を実行できるようにします。

期間: 300

アラームを発生させるデータポイント数: 2

評価期間数: 2

比較演算子: GREATER_THAN_THRESHOLD

AccountProvisionedWriteCapacityUtilization

ディメンション: なし

アラームの説明: このアラームは、アカウントの書き込み容量がプロビジョニングされた制限に達したかどうかを検出します。このような場合は、書き込み容量使用量のアカウントクォータを引き上げることができます。Service Quotas を使用して、書き込み容量ユニットの現在のクォータを表示し、引き上げをリクエストできます。

目的: このアラームは、アカウントの書き込み容量使用率がプロビジョニングされた書き込み容量使用率に近づいているかどうかを検出できます。使用率が上限に達すると、DynamoDB は書き込みリクエストのスロットリングを開始します。

統計: Maximum

推奨しきい値: 80.0

しきい値の根拠: スロットリングを回避するために、しきい値を 80% に設定して、容量が満杯になる前にアクション (アカウント制限の引き上げなど) を実行できるようにします。

期間: 300

アラームを発生させるデータポイント数: 2

評価期間数: 2

比較演算子: GREATER_THAN_THRESHOLD

AgeOfOldestUnreplicatedRecord

ディメンション: TableName、DelegatedOperation

アラームの説明: このアラームは、Kinesis データストリームへのレプリケーションの遅延を検出します。通常のオペレーションでは、`AgeOfOldestUnreplicatedRecord` は数ミリ秒にしかならないはずです。この数字は、顧客が管理する設定での選択が原因でレプリケーションを試行しても成功しなかった回数に基づいて増加します。顧客が管理する設定がレプリケーションの試行が失敗する原因になる例として、Kinesis データストリーム容量のプロビジョニングが不足していたために過剰なスロットリングにつながった場合や、Kinesis データストリームのアクセスポリシーを手動で更新したために DynamoDB がデータストリームにデータを追加できなくなった場

合が挙げられます。このメトリクスを可能な限り低く保つために、Kinesis データストリーム容量を適切にプロビジョニングし、DynamoDB のアクセス許可が変更されていないことを確認する必要があります。

目的: このアラームは、レプリケーション試行の失敗と、その結果生じる Kinesis データストリームへのレプリケーションの遅延をモニタリングできます。

統計: Maximum

推奨しきい値: 状況によって異なります。

しきい値の根拠: 希望するレプリケーション遅延に従って、しきい値をミリ秒単位で設定してください。この値は、ワークロードの要件と期待されるパフォーマンスによって異なります。

期間: 300

アラームを発生させるデータポイント数: 3

評価期間数: 3

比較演算子: GREATER_THAN_THRESHOLD

FailedToReplicateRecordCount

ディメンション: TableName、DelegatedOperation

アラームの説明: このアラームは、DynamoDB が Kinesis データストリームにレプリケートできなかったレコードの数を検出します。34 KB を超える特定の項目はサイズが拡張されて、Kinesis Data Streams の項目サイズ制限 1MB を超えるデータレコードが変更される場合があります。このサイズの拡張は、34 KB を超えるこれらの項目に多数のブール値や空の属性値が含まれる場合に発生します。ブール値と空の属性値は、DynamoDB に 1 バイトで格納されますが、Kinesis Data Streams レプリケーションで標準 JSON を使用してシリアル化すると、最大 5 バイトまで拡張されます。DynamoDB は、このような変更レコードを Kinesis データストリームにレプリケートできません。DynamoDB は、これらの変更データレコードをスキップし、後続のレコードを自動的にレプリケートします。

目的: このアラームは、Kinesis Data Streams のアイテムサイズ制限のために DynamoDB が Kinesis データストリームにレプリケートできなかったレコードの数をモニタリングできます。

統計: Sum

推奨しきい値: 0.0

しきい値の根拠: DynamoDB がレプリケートできなかったレコードをすべて検出するには、しきい値を 0 に設定します。

期間: 60

アラームを発生させるデータポイント数: 1

評価期間数: 1

比較演算子: GREATER_THAN_THRESHOLD

ReadThrottleEvents

ディメンション: TableName

アラームの説明: このアラームは、DynamoDB テーブルに対してスロットリングされている読み取りリクエストの数が多いかどうかを検出します。問題のトラブルシューティングについては、「Amazon DynamoDB でのスロットリングの問題のトラブルシューティング」を参照してください。

目的: このアラームは、DynamoDB テーブルへの読み取りリクエストの持続的なスロットリングを検出できます。読み取りリクエストのスロットリングが続くと、ワークロードの読み取りオペレーションに悪影響を及ぼし、システム全体の効率を低下させる可能性があります。

統計: Sum

推奨しきい値: 状況によって異なります。

しきい値の根拠: スロットリングの許容レベルを考慮して、DynamoDB テーブルの予想読み取りトラフィックに応じてしきい値を設定します。プロビジョニングが不十分なために一貫したスロットリングが行われていないかどうかをモニタリングすることが重要です。履歴データを分析してアプリケーションのワークロードの許容可能なスロットリングレベルを確認し、しきい値が通常のスロットリングレベルよりも高くなるように調整することもできます。スロットリングされたリクエストは一時的なものなので、アプリケーション側またはサービス側で再試行する必要があります。そのため、しきい値を非常に低く設定すると、アラームの感度が高くなりすぎて、望ましくない状態遷移が発生する可能性があります。

期間: 60

アラームを発生させるデータポイント数: 5

評価期間数: 5

比較演算子: GREATER_THAN_THRESHOLD

ReadThrottleEvents

ディメンション: TableName、GlobalSecondaryIndexName

アラームの説明: このアラームは、DynamoDB テーブルのグローバルセカンダリインデックスに対してスロットリングされている読み取りリクエストの数が多いかどうかを検出します。問題のトラブルシューティングについては、「Amazon DynamoDB でのスロットリングの問題のトラブルシューティング」を参照してください。

目的: このアラームは、DynamoDB テーブルのグローバルセカンダリインデックスへの読み取りリクエストの持続的なスロットリングを検出できます。読み取りリクエストのスロットリングが続くと、ワークロードの読み取りオペレーションに悪影響を及ぼし、システム全体の効率を低下させる可能性があります。

統計: Sum

推奨しきい値: 状況によって異なります。

しきい値の根拠: スロットリングの許容レベルを考慮して、DynamoDB テーブルの予想読み取りトラフィックに応じてしきい値を設定します。プロビジョニングが不十分なために一貫したスロットリングが行われていないかどうかをモニタリングすることが重要です。履歴データを分析してアプリケーションのワークロードの許容可能なスロットリングレベルを確認し、しきい値が通常の許容可能なスロットリングレベルよりも高くなるように調整することもできます。スロットリングされたリクエストは一時的なものなので、アプリケーション側またはサービス側で再試行する必要があります。そのため、しきい値を非常に低く設定すると、アラームの感度が高くなりすぎて、望ましくない状態遷移が発生する可能性があります。

期間: 60

アラームを発生させるデータポイント数: 5

評価期間数: 5

比較演算子: GREATER_THAN_THRESHOLD

ReplicationLatency

ディメンション: TableName、ReceivingRegion

アラームの説明: このアラームは、グローバルテーブルのリージョン内のレプリカがソースリージョンより遅れているかどうかを検出します。レイテンシーは、AWS リージョンのパフォーマ

ンスが低下し、そのリージョンにレプリカテーブルが含まれる場合は増加することがあります。この場合、アプリケーションの読み込みおよび書き込みアクティビティを別の AWS リージョンに一時的にリダイレクトすることができます。グローバルテーブルの 2017.11.29 (レガシー) を使用している場合は、書き込み容量ユニット (WCU) が各レプリカテーブルについて同一であることを確認してください。また、「キャパシティを管理するためのベストプラクティスと要件」の推奨事項に必ず従ってください。

目的: このアラームは、あるリージョンのレプリカテーブルで別のリージョンからの変更のレプリケーションに遅延が発生しているかどうかを検出できます。これにより、使用しているレプリカが他のレプリカと異なる可能性があります。各 AWS リージョンのレプリケーションレイテンシーを把握し、そのレプリケーションレイテンシーが継続的に増加する場合はアラートを出すと便利です。テーブルのレプリケーションはグローバルテーブルにのみ適用されます。

統計: Average

推奨しきい値: 状況によって異なります。

しきい値の根拠: このアラームの推奨しきい値は、ユースケースによって大きく異なります。通常、レプリケーションのレイテンシーが 3 分を超える場合は調査が必要です。レプリケーション遅延の重大性と要件を確認し、過去の傾向を分析し、それに応じてしきい値を選択します。

期間: 60

アラームを発生させるデータポイント数: 15

評価期間数: 15

比較演算子: GREATER_THAN_THRESHOLD

SuccessfulRequestLatency

ディメンション: TableName、Operation

アラームの説明: このアラームは、DynamoDB テーブルオペレーション (アラームの `Operation` ディメンション値で示されます) のレイテンシーが大きいことを検出します。Amazon DynamoDB のレイテンシー問題のトラブルシューティングについては、このトラブルシューティングドキュメントを参照してください。

目的: このアラームは、DynamoDB テーブルオペレーションのレイテンシーが大きいことを検出できます。オペレーションのレイテンシーが大きくなると、システム全体の効率に悪影響を及ぼす可能性があります。

統計: Average

推奨しきい値: 状況によって異なります。

しきい値の根拠: DynamoDB では、GetItem や PutItem などのシングルトンオペレーションのレイテンシーは平均 1 桁のミリ秒です。ただし、ワークロードに関係するオペレーションの種類やテーブルについてのレイテンシーの許容範囲に基づいてしきい値を設定できます。このメトリクスの履歴データを分析してテーブルオペレーションの通常のレイテンシーを調べ、しきい値を、オペレーションの重大な遅延を表す数値に設定できます。

期間: 60

アラームを発生させるデータポイント数: 10

評価期間数: 10

比較演算子: GREATER_THAN_THRESHOLD

SystemErrors

ディメンション: TableName

アラームの説明: このアラームは、DynamoDB テーブルリクエストのシステムエラーが継続的に多数発生していることを検出します。引き続き 5xx エラーが表示される場合は、AWS サービスヘルスダッシュボードを開いて、サービスの運用上の問題がないか確認してください。このアラームを使用すると、DynamoDB からの内部サービスの問題が長期にわたって発生した場合に通知を受け取ることができ、クライアントアプリケーションが直面している問題と関連付けるのに役立ちます。詳細については、「DynamoDB でのエラー処理」を参照してください。

目的: このアラームは、DynamoDB テーブルリクエストの持続的なシステムエラーを検出できます。システムエラーは、DynamoDB の内部サービスエラーを示し、クライアントが抱えている問題と関連付けるのに役立ちます。

統計: Sum

推奨しきい値: 状況によって異なります。

しきい値の根拠: システムエラーの許容レベルを考慮して、予想されるトラフィックに応じてしきい値を設定します。また、履歴データを分析してアプリケーションワークロードの許容可能なエラー数を見つけ、それに応じてしきい値を調整することもできます。システムエラーは一時的なものなので、アプリケーション/サービス側で再試行する必要があります。そのため、しきい値

を非常に低く設定すると、アラームの感度が高くなりすぎて、望ましくない状態遷移が発生する可能性があります。

期間: 60

アラームを発生させるデータポイント数: 15

評価期間数: 15

比較演算子: GREATER_THAN_THRESHOLD

ThrottledPutRecordCount

ディメンション: TableName、DelegatedOperation

アラームの説明: このアラームは、変更データキャプチャの Kinesis へのレプリケーションの際に Kinesis データストリームによってスロットリングされるレコードを検出します。このスロットリングは、Kinesis データストリームの容量が不十分であるために発生します。過剰で定期的なスロットリングが発生した場合は、テーブルで観測された書き込みスループットに比例して Kinesis ストリーミングシャードの数を増やす必要があります。Kinesis Data Streams のサイズ決定の詳細については、「Kinesis Data Streams の初期サイズの決定」を参照してください。

目的: このアラームは、Kinesis データストリームの容量が不足しているために、Kinesis データストリームによってスロットリングされたレコードの数をモニタリングできます。

統計: Maximum

推奨しきい値: 状況によって異なります。

しきい値の根拠: 例外的な使用率のピークではスロットリングが発生する可能性がありますが、スロットリングされるレコードを可能な限り少なくしてレプリケーションのレイテンシーを抑える必要があります (DynamoDB は、スロットリングされたレコードの Kinesis データストリームへの送信を再試行します)。しきい値には、過剰なスロットリングが定常的に発生していることを把握するのに役立つ数値を設定します。また、このメトリクスの履歴データを分析して、アプリケーションワークロードの許容可能なスロットリング率を調べることもできます。しきい値は、ユースケースに基づいてアプリケーションで許容可能な値に調整します。

期間: 60

アラームを発生させるデータポイント数: 10

評価期間数: 10

比較演算子: GREATER_THAN_THRESHOLD

UserErrors

ディメンション: なし

アラームの説明: このアラームは、DynamoDB テーブルリクエストのユーザーエラーが継続的に多数発生していることを検出します。問題が発生している期間中にクライアントアプリケーションログをチェックして、リクエストが無効である理由を確認できます。HTTP ステータスコード 400 をチェックして発生しているエラーの種類を確認し、それに応じてアクションを実行できます。有効なリクエストを作成するには、アプリケーションロジックを修正しなければならない場合があります。

目的: このアラームは、DynamoDB テーブルリクエストの持続的なユーザーエラーを検出できます。リクエストされたオペレーションのユーザーエラーは、クライアントが無効なリクエストを生成していて失敗していることを意味します。

統計: Sum

推奨しきい値: 状況によって異なります。

しきい値の根拠: クライアント側のエラーを検出するには、しきい値をゼロに設定します。一方、エラーの数が非常に少ない場合にはアラームがトリガーされないようにしたい場合は、この値をゼロより大きく設定することもできます。ユースケースとリクエストのトラフィックに基づいて決定します。

期間: 60

アラームを発生させるデータポイント数: 10

評価期間数: 10

比較演算子: GREATER_THAN_THRESHOLD

WriteThrottleEvents

ディメンション: TableName

アラームの説明: このアラームは、DynamoDB テーブルに対してスロットリングされている書き込みリクエストの数が多いかどうかを検出します。問題のトラブルシューティングについては、「Amazon DynamoDB でのスロットリングの問題のトラブルシューティング」を参照してください。

目的: このアラームは、DynamoDB テーブルへの書き込みリクエストの持続的なスロットリングを検出できます。書き込みリクエストのスロットリングが続くと、ワークロードの書き込みオペレーションに悪影響を及ぼし、システム全体の効率を低下させる可能性があります。

統計: Sum

推奨しきい値: 状況によって異なります。

しきい値の根拠: スロットリングの許容レベルを考慮して、DynamoDB テーブルの予想書き込みトラフィックに応じてしきい値を設定します。プロビジョニングが不十分なために一貫したスロットリングが行われていないかどうかをモニタリングすることが重要です。履歴データを分析してアプリケーションのワークロードの許容可能なスロットリングレベルを確認し、しきい値が通常の許容可能なスロットリングレベルよりも高い値に調整することもできます。スロットリングされたリクエストは一時的なものなので、アプリケーション/サービス側で再試行する必要があります。そのため、しきい値を非常に低く設定すると、アラームの感度が高くなりすぎて、望ましくない状態遷移が発生する可能性があります。

期間: 60

アラームを発生させるデータポイント数: 5

評価期間数: 5

比較演算子: GREATER_THAN_THRESHOLD

WriteThrottleEvents

ディメンション: TableName、GlobalSecondaryIndexName

アラームの説明: このアラームは、DynamoDB テーブルのグローバルセカンダリインデックスに対してスロットリングされている書き込みリクエストの数が多いかどうかを検出します。問題のトラブルシューティングについては、「Amazon DynamoDB でのスロットリングの問題のトラブルシューティング」を参照してください。

目的: このアラームは、DynamoDB テーブルのグローバルセカンダリインデックスへの書き込みリクエストの持続的なスロットリングを検出できます。書き込みリクエストのスロットリングが続くと、ワークロードの書き込みオペレーションに悪影響を及ぼし、システム全体の効率を低下させる可能性があります。

統計: Sum

推奨しきい値: 状況によって異なります。

しきい値の根拠: スロットリングの許容レベルを考慮して、DynamoDB テーブルの予想書き込みトラフィックに応じてしきい値を設定します。プロビジョニングが不十分なために一貫したスロットリングが行われていないかどうかをモニタリングすることが重要です。履歴データを分析してアプリケーションのワークロードの許容可能なスロットリングレベルを確認し、しきい値が通常の許容可能なスロットリングレベルよりも高い値に調整することもできます。スロットリングされたリクエストは一時的なものなので、アプリケーション/サービス側で再試行する必要があります。そのため、値を非常に低く設定すると、アラームの感度が高くなりすぎて、望ましくない状態遷移が発生する可能性があります。

期間: 60

アラームを発生させるデータポイント数: 5

評価期間数: 5

比較演算子: GREATER_THAN_THRESHOLD

Amazon EBS

VolumeStalledIOCheck

ディメンション: VolumeId、InstanceId

アラームの説明: このアラームは、Amazon EBS ボリュームの IO パフォーマンスをモニタリングするのに役立ちます。このチェックは、Amazon EBS ボリュームの基盤となるストレージサブシステムのハードウェアまたはソフトウェアの問題、Amazon EC2 インスタンスからの Amazon EBS ボリュームの到達可能性に影響する物理ホストのハードウェアの問題など、Amazon EBS インフラストラクチャの根本的な問題を検出するほか、インスタンスと Amazon EBS ボリューム間の接続の問題も検出できます。Stalled IO Check が失敗した場合は、AWS が問題を解決するまで待つか、影響を受けたボリュームの置き換えやボリュームがアタッチされているインスタンスの停止および再起動などのアクションを実行することができます。ほとんどの場合、このメトリクスが失敗すると、Amazon EBS は数分以内にボリュームを自動的に診断して復元します。

目的: このアラームは、Amazon EBS ボリュームのステータスを検出して、これらのボリュームに障害が発生し、I/O 操作を完了できないタイミングを判断できます。

統計: Maximum

推奨しきい値: 1.0

しきい値の根拠: ステータスチェックが不合格になると、このメトリクスの値は 1 になります。しきい値は、ステータスチェックが不合格になるたびにアラームが ALARM 状態になるように設定されます。

期間: 60

アラームを発生させるデータポイント数: 10

評価期間数: 10

比較演算子: GREATER_THAN_OR_EQUAL_TO_THRESHOLD

Amazon EC2

CPUUtilization

ディメンション: InstanceId

アラームの説明: このアラームは EC2 インスタンスの CPU 使用率をモニタリングするのに役立ちます。アプリケーションによっては、使用率が定常的に高いことが普通かもしれません。しかし、パフォーマンスが低下し、アプリケーションがディスク I/O、メモリ、ネットワークリソースの制約を受けていない場合、CPU が上限に達しているということは、リソースのボトルネックやアプリケーションのパフォーマンス上の問題を示している可能性があります。CPU 使用率が高い場合は、CPU 集約度の高いインスタンスへのアップグレードが必要であることを示している可能性があります。詳細モニタリングが有効になっている場合は、この期間を 300 秒から 60 秒に変更できます。詳細については、「インスタンスの詳細モニタリングを有効または無効にする」を参照してください。

目的: このアラームは CPU 使用率が高いことを検出するために使用されます。

統計: Average

推奨しきい値: 80.0

しきい値の根拠: 通常、CPU 使用率のしきい値は 70～80% に設定できます。ただし、許容できるパフォーマンスレベルとワークロード特性に基づいてこの値を調整できます。システムによっては、CPU 使用率が定常的に高いことが正常で問題がない場合もあれば、懸念される事態の原因となる場合もあります。CPU 使用率の履歴データを分析して使用状況を特定し、システムにとって許容できる CPU 使用率を確認し、それに応じてしきい値を設定します。

期間: 300

アラームを発生させるデータポイント数: 3

評価期間数: 3

比較演算子: GREATER_THAN_THRESHOLD

StatusCheckFailed

ディメンション: InstanceId

アラームの説明: このアラームは、システムステータスチェックとインスタンスステータスチェックの両方のモニタリングに役立ちます。いずれかのタイプのステータスチェックが不合格の場合、このアラームは ALARM 状態になります。

目的: このアラームは、システムステータスチェックエラーとインスタンスステータスチェックエラーの両方を含む、インスタンスの根本的な問題を検出するために使用されます。

統計: Maximum

推奨しきい値: 1.0

しきい値の根拠: ステータスチェックが不合格になると、このメトリクスの値は 1 になります。しきい値は、ステータスチェックが不合格になるたびにアラームが ALARM 状態になるように設定されます。

期間: 300

アラームを発生させるデータポイント数: 2

評価期間数: 2

比較演算子: GREATER_THAN_OR_EQUAL_TO_THRESHOLD

StatusCheckFailed_AttachedEBS

ディメンション: InstanceId

アラームの説明: このアラームは、インスタンスにアタッチされている Amazon EBS ボリュームに到達可能かどうか、および I/O 操作を完了できるかどうかをモニタリングするのに役立ちます。このステータスチェックでは、コンピューティングまたは Amazon EBS インフラストラクチャに関する次のような根本的な問題が検出されます。

- Amazon EBS ボリュームの基盤となるストレージサブシステムのハードウェアまたはソフトウェアの問題

- Amazon EBS ボリュームの到達可能性に影響する、物理ホスト上のハードウェアの問題
- インスタンスと Amazon EBS ボリューム間の接続に関する問題

アタッチ済みの EBS ステータスチェックが失敗した場合は、Amazon が問題を解決するまで待つか、影響を受けたボリュームの置き換えやインスタンスの停止および再起動などのアクションを取ることができます。

目的: このアラームは、インスタンスにアタッチされた到達不能な Amazon EBS ボリュームを検出するために使用されます。これにより、I/O 操作が失敗する可能性があります。

統計: Maximum

推奨しきい値: 1.0

しきい値の根拠: ステータスチェックが不合格になると、このメトリクスの値は 1 になります。しきい値は、ステータスチェックが不合格になるたびにアラームが ALARM 状態になるように設定されます。

期間: 60

アラームを発生させるデータポイント数: 10

評価期間数: 10

比較演算子: GREATER_THAN_OR_EQUAL_TO_THRESHOLD

Amazon ElastiCache

CPUUtilization

ディメンション: CacheClusterId、CacheNodeId

アラームの説明: このアラームは、データベースエンジンプロセスやインスタンスで実行されている他のプロセスを含む、ElastiCache インスタンス全体の CPU 使用率のモニタリングに役立ちます。AWSElastiCache は、Memcached と Redis の 2 つのエンジンタイプをサポートしています。Memcached ノードの CPU 使用率が高くなったら、インスタンスタイプをスケールアップするか、新しいキャッシュノードを追加することを検討する必要があります。Redis の場合、主なワークロードが読み取りリクエストによるものである場合は、キャッシュクラスターにリードレプリカをさらに追加することを検討する必要があります。主なワークロードが書き込みリクエ

ストによるものである場合、クラスターモードで実行している場合はシャードを追加してワークロードをより多くのプライマリノードに分散することを検討し、Redis を非クラスターモードで実行している場合はインスタンスタイプをスケールアップすることを検討する必要があります。

目的: このアラームは ElastiCache ホストの CPU 使用率が高いことを検出するために使用されます。エンジン以外のプロセスを含め、インスタンス全体の CPU 使用率を幅広く把握できると便利です。

統計: Average

推奨しきい値: 状況によって異なります。

しきい値の根拠: アプリケーションの CPU 使用率の限界レベルを反映するパーセンテージにしきい値を設定します。Memcached の場合、エンジンは最大 num_threads の個数のコアを使用できます。Redis の場合、エンジンは主にシングルスレッドですが、可能な場合は I/O を高速化するために追加のコアを使用する場合があり、ほとんどの場合、しきい値は使用可能な CPU の約 90% に設定できます。Redis はシングルスレットであるため、実際のしきい値はノードの総容量の割合として計算します。

期間: 60

アラームを発生させるデータポイント数: 5

評価期間数: 5

比較演算子: GREATER_THAN_THRESHOLD

CurrConnections

ディメンション: CacheClusterId、CacheNodeId

アラームの説明: このアラームは接続数が多いことを検出します。これは、負荷が高いか、パフォーマンスに問題があることを示している可能性があります。`CurrConnections` が絶えず増加すると、使用可能な 65,000 の接続を使い果たす可能性があります。アプリケーション側で接続が不適切に閉じられ、サーバー側で確立されたままになっている可能性があります。接続プールまたはアイドル接続タイムアウトを使用してクラスターへの接続数を制限することを検討してください。また、Redis の場合は、クラスターの tcp-keepalive をチューニングして、使用できなくなった可能性のあるピアを検出して終了することを検討してください。

目的: このアラームは、ElastiCache クラスターのパフォーマンスと安定性に影響を与える可能性のある接続数が多い状況を特定するのに役立ちます。

統計: Average

推奨しきい値: 状況によって異なります。

しきい値の根拠: このアラームの推奨しきい値は、クラスターの接続の許容範囲によって大きく異なります。ElastiCache クラスターの容量と予想されるワークロードを確認し、通常使用中の接続数の履歴を分析してベースラインを確立し、それに応じてしきい値を選択します。各ノードは最大 65,000 しか同時に接続をサポートできないことに注意してください。

期間: 60

アラームを発生させるデータポイント数: 10

評価期間数: 10

比較演算子: GREATER_THAN_THRESHOLD

DatabaseMemoryUsagePercentage

ディメンション: CacheClusterId

アラームの説明: このアラームは、クラスターのメモリ使用率をモニタリングするのに役立ちます。`DatabaseMemoryUsagePercentage` が 100% に達すると、Redis maxmemory ポリシーがトリガーされ、選択したポリシーに基づいてエビクションが行われることがあります。キャッシュ内のオブジェクトでエビクションポリシーに一致するものがない場合、書き込みオペレーションは失敗します。一部のワークロードはエビクションを想定していたり、エビクションに依存していたりしますが、エビクションがない場合は、クラスターのメモリ容量を増やす必要があります。プライマリノードを追加してクラスターをスケールアウトすることも、より大きなノードタイプを使用してクラスターをスケールアップすることもできます。詳細については、「ElastiCache for Redis クラスターのスケーリング」を参照してください。

目的: このアラームは、クラスターのメモリ使用率が高いことを検出し、クラスターへの書き込み時に障害が発生するのを防ぐために使用されます。アプリケーションでエビクションが発生しないことが予想される場合、クラスターをスケールアップする必要があるタイミングがわかると便利です。

統計: Average

推奨しきい値: 状況によって異なります。

しきい値の根拠: アプリケーションのメモリ要件と ElastiCache クラスターのメモリ容量に応じて、クラスターのメモリ使用量の限界レベルを反映するパーセンテージにしきい値を設定する必

要があります。メモリ使用量の履歴データを参照して、許容可能なメモリ使用量のしきい値を求めることができます。

期間: 60

アラームを発生させるデータポイント数: 5

評価期間数: 5

比較演算子: GREATER_THAN_THRESHOLD

EngineCPUUtilization

ディメンション: CacheClusterId

アラームの説明: このアラームは ElastiCache インスタンス内の Redis エンジンスレッドの CPU 使用率をモニタリングするのに役立ちます。エンジン CPU 使用率が高くなる一般的な理由としては、長時間実行されるコマンドで高い CPU 使用率が消費されること、多数のリクエスト、短期間での新しいクライアント接続リクエストの増加、キャッシュに新しいデータを保持するための十分なメモリがない場合のエビクションの多さなどが挙げられます。ノードを追加するか、インスタンスタイプをスケールアップして ElastiCache for Redis クラスターのスケーリングをすることを検討してください。

目的: このアラームは Redis エンジンスレッドの CPU 使用率が高いことを検出するために使用されます。データベースエンジン自体の CPU 使用率をモニタリングしたい場合に便利です。

統計: Average

推奨しきい値: 90.0

しきい値の根拠: アプリケーションのエンジン CPU 使用率の限界レベルを反映するパーセンテージにしきい値を設定します。アプリケーションと想定されるワークロードを使用してクラスターのベンチマークを行い、EngineCPUUtilization を基準となるパフォーマンスと相関させ、それに応じてしきい値を設定できます。ほとんどの場合、しきい値は使用可能な CPU の約 90% に設定できます。

期間: 60

アラームを発生させるデータポイント数: 5

評価期間数: 5

比較演算子: GREATER_THAN_THRESHOLD

ReplicationLag

ディメンション: CacheClusterId

アラームの説明: このアラームは、ElastiCache クラスターのレプリケーションの健全性をモニタリングするのに役立ちます。レプリケーションの遅延が大きいということは、プライマリノードまたはレプリカがレプリケーションのペースに追いついていないということです。書き込みアクティビティが多すぎる場合は、プライマリノードを追加してクラスターをスケールアウトするか、より大きなノードタイプを使用してクラスターをスケールアップすることを検討してください。詳細については、「ElastiCache for Redis クラスターのスケーリング」を参照してください。リードレプリカが読み取りリクエストの量によって過負荷になっている場合は、リードレプリカをさらに追加することを検討してください。

目的: このアラームは、プライマリノードでのデータ更新とレプリカノードへの同期の間の遅延を検出するために使用されます。リードレプリカクラスターノードのデータ整合性を確保するのに役立ちます。

統計: Average

推奨しきい値: 状況によって異なります。

しきい値の根拠: アプリケーションの要件とレプリケーション遅延の潜在的な影響に応じてしきい値を設定してください。許容できるレプリケーション遅延については、アプリケーションの予想書き込み速度とネットワークの状態を考慮する必要があります。

期間: 60

アラームを発生させるデータポイント数: 15

評価期間数: 15

比較演算子: GREATER_THAN_THRESHOLD

Amazon EC2 (**AWS/ElasticGPUs**)

GPUConnectivityCheckFailed

ディメンション: InstanceId、EGPUId

アラームの説明: このアラームは、インスタンスと Elastic Graphics アクセラレータ間の接続障害を検出するのに役立ちます。Elastic Graphics は、インスタンスネットワークを使用してリ

モートでアタッチされたグラフィックカードに OpenGL コマンドを送信します。また、Elastic Graphics アクセラレータで OpenGL アプリケーションを実行しているデスクトップは通常、リモートアクセステクノロジーを使用してアクセスされます。OpenGL レンダリングに関連するパフォーマンスの問題とデスクトップのリモートアクセステクノロジーを区別することは重要です。この問題の詳細については、「アプリケーションのパフォーマンス問題の調査」を参照してください。

目的: このアラームは、インスタンスから Elastic Graphics アクセラレータへの接続の問題を検出するために使用されます。

統計: Maximum

推奨しきい値: 0.0

しきい値の根拠: しきい値が 1 の場合は、接続に障害が発生したことを示します。

期間: 300

アラームを発生させるデータポイント数: 3

評価期間数: 3

比較演算子: GREATER_THAN_THRESHOLD

GPUHealthCheckFailed

ディメンション: InstanceId、EGPUId

アラームの説明: このアラームは、Elastic Graphics アクセラレータのステータスに異常があることを知るのに役立ちます。アクセラレータが正常でない場合は、「異常状態の問題の解決」のトラブルシューティング手順を参照してください。

目的: このアラームは、Elastic Graphics アクセラレータが正常でないかどうかを検出するために使用されます。

統計: Maximum

推奨しきい値: 0.0

しきい値の根拠: しきい値が 1 の場合、ステータスチェックが不合格であることを示します。

期間: 300

アラームを発生させるデータポイント数: 3

評価期間数: 3

比較演算子: GREATER_THAN_THRESHOLD

Amazon ECS

CPUReservation

ディメンション: ClusterName

アラームの説明: このアラームは、ECS クラスターの CPU 予約率が高いことを検出するのに役立ちます。CPU 予約率が高い場合は、そのタスク用に登録されている CPU がクラスターに不足している可能性があります。トラブルシューティングのために、容量を追加したり、クラスターをスケールしたり、自動スケーリングを設定したりできます。

目的: このアラームは、クラスター上のタスクによって予約されている CPU ユニットの総数が、クラスターに登録されている CPU ユニットの総数に達しようとしているかどうかを検出するために使用されます。これにより、クラスターをいつスケールアップすべきかがわかります。クラスターの CPU ユニットの合計に達すると、タスクの CPU が不足する可能性があります。EC2 キャパシティプロバイダーのマネージドスケーリングをオンにしている場合、またはキャパシティプロバイダーに Fargate を関連付けている場合、このアラームはお勧めしません。

統計: Average

推奨しきい値: 90.0

しきい値の根拠: CPU 予約のしきい値を 90% に設定します。ただし、クラスターの特性に基づいて、それより低い値を選択できます。

期間: 60

アラームを発生させるデータポイント数: 5

評価期間数: 5

比較演算子: GREATER_THAN_THRESHOLD

CPUUtilization

ディメンション: ClusterName、ServiceName

アラームの説明: このアラームは、ECS サービスの CPU 使用率が高いことを検出するのに役立ちます。ECS のデプロイが進行中でない場合、CPU 使用率が上限に達していると、リソースのボトルネックやアプリケーションのパフォーマンス問題を示している可能性があります。トラブルシューティングのために、CPU 上限を増やすことができます。

目的: このアラームは ECS サービスの CPU 使用率が高いことを検出するために使用されます。CPU 使用率が定常的に高い場合は、リソースのボトルネックやアプリケーションのパフォーマンス問題を示している可能性があります。

統計: Average

推奨しきい値: 90.0

しきい値の根拠: CPU 使用率のサービスメトリクスは 100% の使用率を超える可能性があります。ただし、他のサービスに影響が出ないように、サービスメトリクスの CPU 使用率をモニタリングすることをお勧めします。しきい値は約 90～95% に設定します。他のサービスで今後問題が発生しないように、実際の使用状況を反映するようにタスク定義を更新することをお勧めします。

期間: 60

アラームを発生させるデータポイント数: 5

評価期間数: 5

比較演算子: GREATER_THAN_THRESHOLD

MemoryReservation

ディメンション: ClusterName

アラームの説明: このアラームは、ECS クラスターのメモリ予約率が高いことを検出するのに役立ちます。メモリ予約率が高い場合は、クラスターのリソースボトルネックを示している可能性があります。トラブルシューティングを行うには、サービスタスクのパフォーマンスを分析して、タスクのメモリ使用率を最適化できるかどうかを確認します。また、より多くのメモリを登録したり、自動スケーリングを設定したりできます。

目的: このアラームは、クラスター上のタスクによって予約されているメモリユニットの合計が、クラスターに登録されているメモリユニットの合計に達しようとしているかどうかを検出するために使用されます。これにより、クラスターをいつスケールアップすべきかがわかります。クラスターの合計メモリユニット数に達すると、クラスターは新しいタスクを起動できなくなる

可能性があります。EC2 キャパシティプロバイダーのマネージドスケーリングをオンにしている場合、またはキャパシティプロバイダーに Fargate を関連付けている場合、このアラームはお勧めしません。

統計: Average

推奨しきい値: 90.0

しきい値の根拠: メモリ予約のしきい値を 90% に設定します。これは、クラスターの特性に基づいて、それより低い値に調整できます。

期間: 60

アラームを発生させるデータポイント数: 5

評価期間数: 5

比較演算子: GREATER_THAN_THRESHOLD

HTTPCode_Target_5XX_Count

ディメンション: ClusterName、ServiceName

アラームの説明: このアラームは、ECS サービスのサーバー側エラー数が多いことを検出するのに役立ちます。これは、サーバーがリクエストを処理できない原因となるエラーがあることを示している可能性があります。トラブルシューティングを行うには、アプリケーションログを確認してください。

目的: このアラームは、ECS サービスのサーバー側エラー数が多いことを検出するために使用されます。

統計: Sum

推奨しきい値: 状況によって異なります。

しきい値の根拠: 平均トラフィックの約 5% の値を計算し、この値をしきい値の初期値として使用します。平均トラフィックは、RequestCount メトリクスを使用して求めることができます。また、履歴データを分析してアプリケーションワークロードの許容エラー発生率を判断し、それに応じてしきい値を調整することもできます。頻繁に発生する 5XX エラーにはアラームが必要です。ただし、しきい値を非常に低く設定すると、アラームの感度が高くなりすぎる可能性があります。

期間: 60

アラームを発生させるデータポイント数: 5

評価期間数: 5

比較演算子: GREATER_THAN_THRESHOLD

TargetResponseTime

ディメンション: ClusterName、ServiceName

アラームの説明: このアラームは、ECS サービスリクエストのターゲット応答時間が長いことを検出するのに役立ちます。これは、サービスがリクエストを時間通りに処理できない原因となる問題があることを示している可能性があります。トラブルシューティングを行うには、CPUUtilization メトリクスをチェックしてサービスの CPU が不足していないかどうかを確認するか、サービスが依存する他のダウンストリームサービスの CPU 使用率をチェックします。

目的: このアラームは、ECS サービスリクエストのターゲット応答時間が長いことを検出するために使用されます。

統計: Average

推奨しきい値: 状況によって異なります。

しきい値の根拠: このアラームの推奨しきい値は、ユースケースによって大きく異なります。サービスのターゲット応答時間の重要度と要件を確認し、このメトリクスの過去の動作を分析して、適切なしきい値レベルを決定します。

期間: 60

アラームを発生させるデータポイント数: 5

評価期間数: 5

比較演算子: GREATER_THAN_THRESHOLD

Container Insights を使用する Amazon ECS

EphemeralStorageUtilized

ディメンション: ClusterName、ServiceName

アラームの説明: このアラームは、Fargate クラスターのエフェメラルストレージの使用率が高いことを検出するのに役立ちます。エフェメラルストレージの使用率が常に高い場合は、エフェメラルストレージの使用状況を確認して、エフェメラルストレージを増やすことができます。

目的: このアラームは、Fargate クラスターのエフェメラルストレージ使用率が高いことを検出するために使用されます。エフェメラルストレージの使用率が高い状態が続くと、ディスクがいっぱいになり、コンテナに障害が発生する可能性があります。

統計: Average

推奨しきい値: 状況によって異なります。

しきい値の根拠: しきい値をエフェメラルストレージサイズの約 90% に設定します。この値は、Fargate クラスターの許容可能なエフェメラルストレージ使用率に基づいて調整できます。一部のシステムでは、エフェメラルストレージの使用率が常に高いことが正常である場合もあれば、コンテナの障害につながる場合もあります。

期間: 60

アラームを発生させるデータポイント数: 5

評価期間数: 5

比較演算子: GREATER_THAN_THRESHOLD

RunningTaskCount

ディメンション: ClusterName、ServiceName

アラームの説明: このアラームは、ECS サービスの実行中のタスク数が少ないことを検出するのに役立ちます。実行中のタスク数が少なすぎる場合は、アプリケーションがサービスの負荷を処理できず、パフォーマンスの問題が発生する可能性があります。実行中のタスクがない場合は、Amazon ECS サービスが利用できないか、デプロイに問題がある可能性があります。

目的: このアラームは、実行中のタスクの数が少なすぎるかどうかを検出するために使用されます。実行中のタスク数が常に少ない場合は、ECS サービスのデプロイやパフォーマンスに問題がある可能性があります。

統計: Average

推奨しきい値: 0.0

しきい値の根拠: ECS サービスの実行中の最小タスク数に基づいてしきい値を調整できます。実行中のタスク数が 0 の場合、Amazon ECS サービスは利用できません。

期間: 60

アラームを発生させるデータポイント数: 5

評価期間数: 5

比較演算子: LESS_THAN_OR_EQUAL_TO_THRESHOLD

instance_filesystem_utilization

ディメンション: InstanceId、ContainerInstanceId、ClusterName

アラームの説明: このアラームは、ECS クラスターのファイルシステムの使用率が高いことを検出するのに役立ちます。ファイルシステムの使用率が常に高い場合は、ディスク使用量を確認してください。

目的: このアラームは、Amazon ECS クラスターのファイルシステムの使用率が高いことを検出するために使用されます。ファイルシステムの使用率が常に高い場合は、リソースのボトルネックやアプリケーションのパフォーマンスの問題が発生している可能性があり、新しいタスクを実行できなくなる可能性があります。

統計: Average

推奨しきい値: 90.0

しきい値の根拠: ファイルシステム使用率のしきい値は約 90～95% に設定できます。この値は、Amazon ECS クラスターの許容可能なファイルシステム容量レベルに基づいて調整できます。一部のシステムでは、ファイルシステムの使用率が常に高いことが正常で問題がない場合もあれば、他のシステムでは、懸念される事態の原因となり、パフォーマンスの問題が発生して、新しいタスクを実行できなくなる場合もあります。

期間: 60

アラームを発生させるデータポイント数: 5

評価期間数: 5

比較演算子: GREATER_THAN_THRESHOLD

Amazon EFS

PercentIOLimit

ディメンション: FileSystemId

アラームの説明: このアラームは、ワークロードがファイルシステムで利用可能な I/O 制限内に収まるようにするのに役立ちます。メトリクスが定常的に I/O 上限に達している場合は、モードとして Max I/O を使用するファイルシステムにアプリケーションを移動することを検討してください。トラブルシューティングのため、ファイルシステムに接続しているクライアントと、ファイルシステムをスロットリングしているクライアントのアプリケーションを確認します。

目的: このアラームは、ファイルシステムが汎用パフォーマンスモードの I/O 制限にどれだけ近づいているかを検出するのに使用されます。I/O の割合が定常的に高いということは、I/O 要求に対してファイルシステムを十分にスケールできないことを示している可能性があり、ファイルシステムを使用するアプリケーションにとってファイルシステムがリソースのボトルネックになる可能性があります。

統計: Average

推奨しきい値: 100.0

しきい値の根拠: ファイルシステムが I/O の上限に達すると、読み取りリクエストや書き込みリクエストへの応答が遅くなる可能性があります。そのため、ファイルシステムを使用するアプリケーションに影響を与えないように、メトリクスをモニタリングすることをお勧めします。しきい値は 100% 程度に設定できます。ただし、この値はファイルシステムの特性に基づいてそれより低い値に調整できます。

期間: 60

アラームを発生させるデータポイント数: 15

評価期間数: 15

比較演算子: GREATER_THAN_OR_EQUAL_TO_THRESHOLD

BurstCreditBalance

ディメンション: FileSystemId

アラームの説明: このアラームは、ファイルシステムの使用に対するバーストクレジットバランスがあることを確認するのに役立ちます。使用可能なバーストクレジットがない場合、スルー

プットが低下するため、ファイルシステムへのアプリケーションのアクセスが制限されます。メトリクスが定常的に 0 に低下する場合は、スループットモードをエラスティックスループットモードまたはプロビジョンドスループットモードに変更することを検討してください。

目的: このアラームは、ファイルシステムのバーストクレジットバランスが少ないことを検出するために使用されます。バーストクレジットバランスが定常的に低いことは、スループットの低下と I/O レイテンシーの増加を示している可能性があります。

統計: Average

推奨しきい値: 0.0

しきい値の根拠: ファイルシステムがバーストクレジットを使い果たすと、ベースラインスループットレートが低くても、EFS はすべてのファイルシステムに対して 1 MiBps の従量制スループットを提供し続けます。ただし、ファイルシステムがアプリケーションのリソースボトルネックになるのを避けるため、バーストクレジットバランスが低いかどうかメトリクスをモニタリングすることをお勧めします。しきい値は 0 バイト程度に設定できます。

期間: 60

アラームを発生させるデータポイント数: 15

評価期間数: 15

比較演算子: LESS_THAN_OR_EQUAL_TO_THRESHOLD

Container Insights を使用する Amazon EKS

node_cpu_utilization

ディメンション: ClusterName

アラームの説明: このアラームは、EKS クラスターのワーカーノードの CPU 使用率が高いことを検出するのに役立ちます。使用率が定常的に高い場合は、ワーカーノードを CPU の大きいインスタンスに置き換えるか、システムを水平的にスケールする必要があることを示している可能性があります。

目的: このアラームは、EKS クラスター内のワーカーノードの CPU 使用率をモニタリングして、システムパフォーマンスが低下しないようにするのに役立ちます。

統計: Maximum

推奨しきい値: 80.0

しきい値の根拠: システムに影響が出始める前に問題をデバッグするのに十分な時間を確保するために、しきい値を 80% 以下に設定することをお勧めします。

期間: 60

アラームを発生させるデータポイント数: 5

評価期間数: 5

比較演算子: GREATER_THAN_THRESHOLD

node_filesystem_utilization

ディメンション: ClusterName

アラームの説明: このアラームは、EKS クラスターのワーカーノードでファイルシステムの使用率が高いことを検出するのに役立ちます。使用率が定常的に高い場合は、ワーカーノードを更新してディスクボリュームを大きくするか、水平的にスケールする必要があることを示している可能性があります。

目的: このアラームは、EKS クラスター内のワーカーノードのファイルシステムの使用率をモニタリングするのに役立ちます。使用率が 100% に達すると、アプリケーション障害、ディスク I/O のボトルネック、ポッドエビクション、またはノードの完全無応答が発生する可能性があります。

統計: Maximum

推奨しきい値: 状況によって異なります。

しきい値の根拠: 十分なディスク負荷がかかっている (つまり、ディスクがいっぱいになっている) 場合、ノードは正常ではないとマークされ、ポッドはノードからエビクションされます。利用可能なファイルシステムが kubelet に設定されたエビクションのしきい値を下回ると、ディスク負荷がかかっているノード上のポッドはエビクションされます。アラームのしきい値を設定して、ノードがクラスターからエビクションされる前に対応する時間が十分にあるようにします。

期間: 60

アラームを発生させるデータポイント数: 5

評価期間数: 5

比較演算子: GREATER_THAN_THRESHOLD

node_memory_utilization

ディメンション: ClusterName

アラームの説明: このアラームは、EKS クラスターのワーカーノードのメモリ使用率が高いことを検出するのに役立ちます。使用率が定常的に高い場合は、ポッドレプリカの数をスケールするか、アプリケーションを最適化する必要があることを示している可能性があります。

目的: このアラームは、EKS クラスター内のワーカーノードのメモリ使用率をモニタリングして、システムパフォーマンスが低下しないようにするのに役立ちます。

統計: Maximum

推奨しきい値: 80.0

しきい値の根拠: システムに影響が出始める前に問題をデバッグするのに十分な時間を確保するために、しきい値を 80% 以下に設定することをお勧めします。

期間: 60

アラームを発生させるデータポイント数: 5

評価期間数: 5

比較演算子: GREATER_THAN_THRESHOLD

pod_cpu_utilization_over_pod_limit

ディメンション: ClusterName、Namespace、Service

アラームの説明: このアラームは、EKS クラスターのポッドの CPU 使用率が高いことを検出するのに役立ちます。使用率が定常的に高い場合は、影響を受けるポッドの CPU 上限を増やす必要があることを示している可能性があります。

目的: このアラームは、EKS クラスター内の Kubernetes Service に属するポッドの CPU 使用率をモニタリングするのに役立ちます。これにより、サービスのポッドが CPU を予想より多く消費しているかどうかをすばやく特定できます。

統計: Maximum

推奨しきい値: 80.0

しきい値の根拠: システムに影響が出始める前に問題をデバッグするのに十分な時間を確保するために、しきい値を 80% 以下に設定することをお勧めします。

期間: 60

アラームを発生させるデータポイント数: 5

評価期間数: 5

比較演算子: GREATER_THAN_THRESHOLD

pod_memory_utilization_over_pod_limit

ディメンション: ClusterName、Namespace、Service

アラームの説明: このアラームは、EKS クラスターのポッドのメモリ使用率が高いことを検出するのに役立ちます。使用率が定常的に高い場合は、影響を受けるポッドのメモリ上限を増やす必要があることを示している可能性があります。

目的: このアラームは、EKS クラスター内のポッドのメモリ使用率をモニタリングして、システムパフォーマンスが低下しないようにするのに役立ちます。

統計: Maximum

推奨しきい値: 80.0

しきい値の根拠: システムに影響が出始める前に問題をデバッグするのに十分な時間を確保するために、しきい値を 80% 以下に設定することをお勧めします。

期間: 60

アラームを発生させるデータポイント数: 5

評価期間数: 5

比較演算子: GREATER_THAN_THRESHOLD

Amazon Kinesis Data Streams

GetRecords.IteratorAgeMilliseconds

ディメンション: StreamName

アラームの説明: このアラームは、イテレータの最大存続期間が長すぎるかどうかを検出できます。リアルタイムデータ処理アプリケーションでは、遅延の許容度に従ってデータ保持を設定します。通常は数分以内です。履歴データを処理するアプリケーションでは、このメトリクスを使用してキャッチアップ速度をモニタリングします。問題のトラブルシューティングを行う間、データ損失を防ぐ手っ取り早い解決策は、保存期間を長くすることです。また、コンシューマーアプリケーションのレコードを処理するワーカーの数を増やすこともできます。イテレータの存続期間がしだいに増加する一般的な原因は、物理リソースの不足またはストリームスループットの上昇に応じてレコード処理ロジックがスケールされていないことです。詳細については、リンクを参照してください。

目的: このアラームは、ストリーム内のデータが、保存期間が長すぎるかレコード処理が遅すぎるために期限切れになるかどうかを検出するために使用されます。ストリームの保持期間が 100% に達した後のデータ損失を防ぐのに役立ちます。

統計: Maximum

推奨しきい値: 状況によって異なります。

しきい値の根拠: このアラームの推奨しきい値は、ストリームの保持期間とレコードの処理遅延の許容度に大きく依存します。要件を確認して過去の傾向を分析し、しきい値を重大な処理遅延を表すミリ秒単位の数値に設定します。イテレータの存続期間が保持期間 (デフォルトで 24 時間、最大で 365 日まで設定可能) の 50% を経過すると、レコードの有効期限切れによるデータ損失のリスクがあります。メトリクスをモニタリングして、どのシャードもこの制限に接近したことがないことを確認できます。

期間: 60

アラームを発生させるデータポイント数: 15

評価期間数: 15

比較演算子: GREATER_THAN_THRESHOLD

GetRecords.Success

ディメンション: StreamName

アラームの説明: このメトリクスは、コンシューマーがストリームからデータを正常に読み取るたびに増加します。`GetRecords` は、例外をスローすると、データを何も返しません。最もよくある例外は `ProvisionedThroughputExceededException` です。ストリームのリクエストレートが高すぎることや、利用可能なスループットが既に一定の秒数供給されてしまったことが原因です。リクエストの頻度またはサイズを少なくします。詳細については、「Amazon Kinesis

Data Streams デベロッパーガイド」の「制限」と、「AWS でのエラー再試行とエクスポネンシャルバックオフ」を参照してください。

目的: このアラームは、コンシューマーによるストリームからのレコードの取得が失敗したかどうかを検出できます。このメトリクスにアラームを設定することで、エラー率の上昇や取得の成功率の低下など、データ消費に関する問題を事前に検出できます。これにより、潜在的な問題を解決し、円滑なデータ処理パイプラインを維持するためのアクションをタイムリーに実行できます。

統計: Average

推奨しきい値: 状況によって異なります。

しきい値の根拠: ストリームからレコードを取得することの重要性に応じて、失敗したレコードに対するアプリケーションの許容度に基づいてしきい値を設定します。しきい値は、その許容度に対応する成功オペレーションの割合にする必要があります。GetRecords の過去のメトリクスデータを許容可能な失敗率のリファレンスとして使用できます。また、失敗したレコードは再試行できるため、しきい値を設定する際には再試行も考慮する必要があります。これにより、一時的なスパイクによって不要なアラートがトリガーされるのを防ぐことができます。

期間: 60

アラームを発生させるデータポイント数: 5

評価期間数: 5

比較演算子: LESS_THAN_THRESHOLD

PutRecord.Success

ディメンション: StreamName

アラームの説明: このアラームは、失敗した `PutRecord` オペレーションの数がしきい値を超えたことを検出します。データプロデューサーのログを調べて、失敗の根本原因を見つけます。最もよくある理由は、`ProvisionedThroughputExceededException` の原因となったシャードのプロビジョニングされたスループットが不十分であることです。これは、ストリームのリクエストレートが高すぎるか、シャードに取り込もうとしたスループットが高すぎることが原因です。リクエストの頻度またはサイズを少なくします。詳細については、「制限」と「AWS でのエラーの再試行とエクスポネンシャルバックオフ」を参照してください。

目的: このアラームは、ストリームへのレコードの取り込みが失敗しているかどうかを検出できます。ストリームにデータを書き込む際の問題を特定するのに役立ちます。このメトリクスにア

ラームを設定することで、エラー率の上昇や公開に成功したレコードの減少など、ストリームにデータを公開する際にプロデューサーが抱える問題を事前に検出できます。これにより、潜在的な問題に対処し、信頼性の高いデータインジェストプロセスを維持するためのアクションをタイムリーに実行できます。

統計: Average

推奨しきい値: 状況によって異なります。

しきい値の根拠: サービスへのデータインジェストと処理の重要性に応じて、失敗したレコードに対するアプリケーションの許容度に基づいてしきい値を設定します。しきい値は、その許容度に対応する成功オペレーションの割合にする必要があります。PutRecord の過去のメトリクスデータを許容可能な失敗率のリファレンスとして使用できます。また、失敗したレコードは再試行できるため、しきい値を設定する際には再試行も考慮する必要があります。

期間: 60

アラームを発生させるデータポイント数: 5

評価期間数: 5

比較演算子: LESS_THAN_THRESHOLD

PutRecords.FailedRecords

ディメンション: StreamName

アラームの説明: このアラームは、失敗した PutRecords の数がしきい値を超えたことを検出します。Kinesis Data Streams は、各 PutRecords リクエストのすべてのレコードを処理しようとしますが、1 つのレコードに失敗が発生しても後続のレコードの処理は停止しません。これらの失敗の主な理由は、ストリームまたは個々のシャードのスループットを超えていることです。よくある原因としては、レコードがストリームに不均等に届く原因となるトラフィックスパイクやネットワーク遅延があります。正常に処理されなかったレコードを検出し、それ以降の呼び出しで再試行する必要があります。詳細については、「PutRecords を使用するときのエラーの処理」を参照してください。

目的: このアラームは、バッチオペレーションを使用してストリームにレコードを追加する際に定常的に発生するエラーを検出できます。このメトリクスにアラームを設定することで、失敗したレコードの増加を事前に検出できるため、根本的な問題に対処するためのアクションをタイムリーに実行し、円滑で信頼性の高いデータインジェストプロセスを確保できます。

統計: Sum

推奨しきい値: 状況によって異なります。

しきい値の根拠: 失敗したレコードに対するアプリケーションの許容度を反映して、失敗したレコード数のしきい値を設定します。過去のデータを許容可能な失敗の値のリファレンスとして使用できます。また、失敗したレコードは後続の PutRecords 呼び出しで再試行できるため、しきい値を設定する際には再試行も考慮する必要があります。

期間: 60

アラームを発生させるデータポイント数: 5

評価期間数: 5

比較演算子: GREATER_THAN_THRESHOLD

ReadProvisionedThroughputExceeded

ディメンション: StreamName

アラームの説明: このアラームは、読み取りスループットキャパシティのスロットリングの原因となったレコードの数を追跡します。定常的にスロットリングされていることがわかった場合は、ストリームにシャードを追加して、プロビジョニングされた読み取りスループットを増やすことを検討してください。ストリーム上で実行されているコンシューマーアプリケーションが複数あり、それらが `GetRecords` 制限を共有している場合は、Enhanced Fan-Out を使用して新しいコンシューマーアプリケーションを登録することをお勧めします。シャードを追加してもスロットリングの数が減らない場合は、他のシャードよりも多くの読み取りを受けている「ホット」シャードがある可能性があります。拡張モニタリングを有効にし、「ホット」シャードを見つけて分割します。

目的: このアラームは、コンシューマーがプロビジョニングされた読み取りスループット (所有するシャードの数によって決まる) を超えたときに、コンシューマーがスロットリングされているかどうかを検出できます。その場合、ストリームからの読み取りはできず、ストリームはバックアップを開始する可能性があります。

統計: Average

推奨しきい値: 状況によって異なります。

しきい値の根拠: 通常、スロットリングされたリクエストは再試行できるため、しきい値を 0 に設定するとアラームの感度が高くなりすぎます。ただし、定常的なスロットリングは、ストリームからの読み取りに影響する可能性があるため、アラームをトリガーする必要があります。アプ

リケーションのスロットリングされたリクエストに応じた割合にしきい値を設定し、設定を再試行します。

期間: 60

アラームを発生させるデータポイント数: 5

評価期間数: 5

比較演算子: GREATER_THAN_THRESHOLD

SubscribeToShardEvent.MillisBehindLatest

ディメンション: StreamName、ConsumerName

アラームの説明: このアラームは、アプリケーションのレコード処理の遅延がしきい値を超えたことを検出します。ダウンストリームのアプリケーションで API オペレーションが失敗するなどの一時的な問題により、メトリクスが突然増加する可能性があります。定常的に発生しているかどうかを調査する必要があります。よくある原因は、物理リソースの不足や、レコード処理ロジックがストリームスループットの増加に伴ってスケールしなかったために、コンシューマーがレコードを十分な速度で処理していないことです。クリティカルパス内の呼び出しをブロックすることが、レコード処理の速度低下の原因となることがよくあります。シャードの数を増やして並列処理を増やすことができます。また、需要のピーク時には、基盤となる処理ノードに十分な物理リソースがあることを確認する必要もあります。

目的: このアラームは、ストリームのシャードイベントへのサブスクリプションの遅延を検出できます。これは処理遅延を示しており、コンシューマーアプリケーションのパフォーマンスやストリーム全体の健全性に関する潜在的な問題を特定するのに役立ちます。処理遅延が顕著になった場合は、ボトルネックやコンシューマーアプリケーションの非効率性を調査して対処し、リアルタイムのデータ処理を確保し、データバックログを最小限に抑える必要があります。

統計: Average

推奨しきい値: 状況によって異なります。

しきい値の根拠: このアラームの推奨しきい値は、アプリケーションで許容できる遅延によって大きく異なります。アプリケーションの要件を確認して過去の傾向を分析し、それに応じてしきい値を選択します。SubscribeToShard 呼び出しが成功すると、コンシューマーは持続的接続を介して最大 5 分間の SubscribeToShardEvent イベントの受信を開始します。その後、レコードを引き続き受信したい場合は、SubscribeToShard を再度呼び出してサブスクリプションを更新する必要があります。

期間: 60

アラームを発生させるデータポイント数: 5

評価期間数: 5

比較演算子: GREATER_THAN_THRESHOLD

WriteProvisionedThroughputExceeded

ディメンション: StreamName

アラームの説明: このアラームは、書き込みスループットキャパシティのスロットリングの原因となったレコード数がしきい値に達したことを検出します。プロデューサーがプロビジョニングされた書き込みスループット (所有しているシャードの数によって決まる) を超えると、プロデューサーはスロットリングされ、ストリームにレコードを追加できなくなります。定常的なスロットリングに対処するには、ストリームにシャードを追加することを検討する必要があります。これにより、プロビジョニングされた書き込みスループットが増加し、その後のスロットリングが防止されます。レコードを取り込む際には、パーティションキーの選択も考慮する必要があります。ランダムパーティションキーは、可能な限りストリームのシャード全体にレコードを均等に分散させるため、推奨されます。

目的: このアラームは、ストリームまたはシャードのスロットリングが原因で、プロデューサーがレコードの書き込みを拒否されているかどうかを検出できます。ストリームがプロビジョンドモードの場合、このアラームを設定すると、データストリームが制限に達したときに事前にアクションを実行できるため、データ損失を防ぎ、円滑なデータ処理を維持するために、プロビジョニングされた容量を最適化したり、適切なスケーリングアクションを実行したりできます。

統計: Average

推奨しきい値: 状況によって異なります。

しきい値の根拠: 通常、スロットリングされたリクエストは再試行できるため、しきい値を 0 に設定するとアラームの感度が高くなりすぎます。ただし、定常的なスロットリングは、ストリームへの書き込みに影響する可能性があるため、アラームのしきい値を設定してこれを検出する必要があります。アプリケーションのスロットリングされたリクエストに応じた割合にしきい値を設定し、設定を再試行します。

期間: 60

アラームを発生させるデータポイント数: 5

評価期間数: 5

比較演算子: GREATER_THAN_THRESHOLD

Lambda

ClaimedAccountConcurrency

ディメンション: なし

アラームの説明: このアラームは、Lambda 関数の同時実行数がアカウントのリージョンレベルの同時実行制限に近づいているかどうかをモニタリングするのに役立ちます。同時実行数の上限に達すると、関数はスロットリングされ始めます。以下のアクションを実行して、スロットリングを回避できます。

1. このリージョンで同時実行数の増加をリクエストします。
2. 未使用の予約された同時実行またはプロビジョニングされた同時実行を特定して減らします。
3. 関数内のパフォーマンスの問題を特定して処理速度を向上させ、スループットを向上させます。
4. 関数のバッチサイズを増やして、各関数呼び出しでより多くのメッセージが処理されるようにします。

目的: このアラームは、Lambda 関数の同時実行数がアカウントのリージョンレベルの同時実行クォータに近づいているかどうかを事前に検出し、ユーザーがそれに対応できるようにします。`ClaimedAccountConcurrency` がアカウントのリージョンレベルの同時実行クォータに達すると、関数はスロットリングされます。リザーブド同時実行 (RC) またはプロビジョニングされた同時実行 (PC) を使用している場合、このアラームにより、`ConcurrentExecutions` のアラームよりも同時実行の使用率をより詳細に把握できます。

統計: Maximum

推奨しきい値: 状況によって異なります。

しきい値の根拠: リージョン内のアカウントに設定されている同時実行クォータの約 90% の値を計算し、その結果をしきい値として使用してください。デフォルトで、アカウントの同時実行クォータは 1 つのリージョン内のすべての関数全体で 1,000 です。ただし、Service Quotas ダッシュボードでアカウントのクォータを確認する必要があります。

期間: 60

アラームを発生させるデータポイント数: 10

評価期間数: 10

比較演算子: GREATER_THAN_THRESHOLD

エラー

ディメンション: FunctionName

アラームの説明: このアラームは、エラー数が多いことを検出します。エラーには、コードによってスローされた例外と、Lambda ランタイムによってスローされた例外が含まれます。関数に関連するログを確認して問題を診断できます。

目的: このアラームは、関数呼び出しで発生したエラーの数が多いことを検出するのに役立ちます。

統計: Sum

推奨しきい値: 状況によって異なります。

しきい値の根拠: しきい値を 0 より大きい数値に設定します。正確な値は、アプリケーションのエラー許容度によって異なります。関数が処理する呼び出しの重要度を把握してください。アプリケーションによっては、どのようなエラーも許容できない場合もあれば、ある程度の誤差を許容できるアプリケーションもあります。

期間: 60

アラームを発生させるデータポイント数: 3

評価期間数: 3

比較演算子: GREATER_THAN_THRESHOLD

Throttles

ディメンション: FunctionName

アラームの説明: このアラームは、スロットリングされた呼び出しリクエストが多いことを検出します。スロットリングは、スケールアップに同時実行が利用できない場合に発生します。この問題を解決するには、いくつかのアプローチがあります。1) このリージョンの AWS サポートに同時実行数の増加をリクエストします。2) 関数内のパフォーマンスの問題を特定して処理速度を向上させ、それによってスループットを向上させます。3) 関数のバッチサイズを増やして、関数が呼び出されるたびにより多くのメッセージが処理されるようにします。

目的: このアラームは、Lambda 関数に対するスロットリングされた呼び出しリクエストの数が多いことを検出するのに役立ちます。スロットリングが原因でリクエストが常に拒否されているのかを知ることや、定常的なスロットリングを回避するために Lambda 関数のパフォーマンスを改善したり同時実行容量を増したりする必要があるのかを知ることが重要です。

統計: Sum

推奨しきい値: 状況によって異なります。

しきい値の根拠: しきい値を 0 より大きい数値に設定します。しきい値の正確な値は、アプリケーションの許容度によって異なる場合があります。アプリケーションの用途と関数のスケーリング要件に従ってしきい値を設定します。

期間: 60

アラームを発生させるデータポイント数: 5

評価期間数: 5

比較演算子: GREATER_THAN_OR_EQUAL_TO_THRESHOLD

duration

ディメンション: FunctionName

アラームの説明: このアラームは、Lambda 関数によるイベントの処理時間が長いことを検出します。処理時間が長くなるのは、関数の実行時間が長くなるような関数コードの変更や、関数の依存関係の時間が長くなることが原因の可能性があります。

目的: このアラームは、Lambda 関数の実行時間が長いことを検出できます。実行時間が長いということは、関数の呼び出しに時間がかかることを示し、Lambda が処理するイベントの数が多い場合は、呼び出しの同時実行容量にも影響する可能性があります。Lambda 関数の実行時間が常に予想よりも長くなっているかどうかを知ることは重要です。

統計: p90

推奨しきい値: 状況によって異なります。

しきい値の根拠: 経過時間のしきい値は、アプリケーションとワークロード、およびパフォーマンス要件によって異なります。高性能が要件の場合は、しきい値を短く設定して、その関数が想定どおりかどうかを確認します。また、経過時間メトリクスの履歴データを分析して、かかった時間が関数の想定パフォーマンスと一致するかどうかを確認し、しきい値を過去の平均よりも長

い時間に設定することもできます。しきい値は、設定した関数タイムアウトよりも短く設定してください。

期間: 60

アラームを発生させるデータポイント数: 15

評価期間数: 15

比較演算子: GREATER_THAN_THRESHOLD

ConcurrentExecutions

ディメンション: FunctionName

アラームの説明: このアラームは、関数の同時実行数がアカウントのリージョンレベルの同時実行制限に近づいているかどうかをモニタリングするのに役立ちます。同時実行数の上限に達すると、関数はスロットリングされ始めます。以下のアクションを実行して、スロットリングを回避できます。

1. このリージョンで同時実行数の増加をリクエストします。
2. 関数内のパフォーマンスの問題を特定して処理速度を向上させ、スループットを向上させます。
3. 関数のバッチサイズを増やして、各関数呼び出しでより多くのメッセージが処理されるようにします。

予約された同時実行数とプロビジョニングされた同時実行数の使用率をより明確に把握するには、代わりに新しいメトリクス `ClaimedAccountConcurrency` にアラームを設定します。

目的: このアラームは、関数の同時実行数がアカウントのリージョンレベルの同時実行クォータに近づいているかどうかを事前に検出し、ユーザーが対応できるようにします。アカウントのリージョンレベルの同時実行クォータに達すると、関数はスロットリングされます。

統計: Maximum

推奨しきい値: 状況によって異なります。

しきい値の根拠: リージョンのアカウントに設定されている同時実行クォータの約 90% にしきい値を設定します。デフォルトで、アカウントの同時実行クォータは 1 つのリージョン内のすべての関数全体で 1,000 です。ただし、AWS サポートに連絡すれば増やすことができるため、アカウントのクォータを確認することに意味はあります。

期間: 60

アラームを発生させるデータポイント数: 10

評価期間数: 10

比較演算子: GREATER_THAN_THRESHOLD

Lambda Insights

以下の Lambda Insights メトリクスにベストプラクティスアラームを設定することをお勧めします。

memory_utilization

ディメンション: function_name

アラームの説明: このアラームは、Lambda 関数のメモリ使用率が設定された上限に近づいているかどうかを検出するために使用されます。トラブルシューティングのためにできる試行として、1) コードを最適化します。2) メモリ要件を正確に見積もって、メモリ割り当てのサイズを適切に設定します。同じことについては、「Lambda Power Tuning」を参照することもできます。3) 接続プーリングを使用します。RDS データベースの接続プーリングについては、「Amazon RDS Proxy を Lambda で使用する」を参照してください。4) また、呼び出しと呼び出しの間に大量のデータをメモリに保存しないように関数を設計することも検討できます。

目的: このアラームは、Lambda 関数のメモリ使用率が設定された上限に近づいているかどうかを検出するために使用されます。

統計: Average

推奨しきい値: 90.0

しきい値の根拠: しきい値を 90% に設定して、メモリ使用率が割り当てられたメモリの 90% を超えるとアラートが表示されるようにします。ワークロードのメモリ使用率が気になる場合は、この値をそれより低い値に調整できます。このメトリクスの履歴データを確認して、それに応じてしきい値を設定することもできます。

期間: 60

アラームを発生させるデータポイント数: 10

評価期間数: 10

比較演算子: GREATER_THAN_THRESHOLD

Amazon VPC (**AWS/NATGateway**)

ErrorPortAllocation

ディメンション: NatGatewayId

アラームの説明: このアラームは、NAT ゲートウェイが新しい接続へのポートの割り当てができないことを検出するのに役立ちます。この問題を解決するには、「NAT Gateway でのポートアロケーションエラーを解決する」を参照してください。

目的: このアラームは、NAT ゲートウェイがソースポートの割り当てができなかったかどうかを検出するために使用されます。

統計: Sum

推奨しきい値: 0.0

しきい値の根拠: ErrorPortAllocation の値がゼロより大きい場合、多くの接続を集めている単一の送信先に NAT ゲートウェイを介して開かれている同時接続が多すぎるということです。

期間: 60

アラームを発生させるデータポイント数: 15

評価期間数: 15

比較演算子: GREATER_THAN_THRESHOLD

PacketsDropCount

ディメンション: NatGatewayId

アラームの説明: このアラームは、NAT ゲートウェイによってパケットがドロップされたことを検出するのに役立ちます。これは NAT ゲートウェイの問題が原因で発生する可能性があるため、AWS サービスヘルスダッシュボードで、リージョンの AWS NAT ゲートウェイのステータスを確認してください。これにより、NAT ゲートウェイを使用するトラフィックに関連するネットワークの問題を関連付けることができます。

目的: このアラームは、パケットが NAT ゲートウェイによってドロップされているかどうかを検出するために使用されます。

統計: Sum

推奨しきい値: 状況によって異なります。

しきい値の根拠: NAT ゲートウェイの総トラフィックの 0.01% の値を計算し、その結果をしきい値として使用してください。NAT ゲートウェイのトラフィックの履歴データを使用してしきい値を決定します。

期間: 60

アラームを発生させるデータポイント数: 5

評価期間数: 5

比較演算子: GREATER_THAN_THRESHOLD

AWS プライベートリンク (**AWS/PrivateLinkEndpoints**)

PacketsDropped

ディメンション: VPC Id、VPC Endpoint Id、Endpoint Type、Subnet Id、Service Name

アラームの説明: このアラームは、エンドポイントがドロップしたパケット数をモニタリングすることで、エンドポイントまたはエンドポイントサービスに異常があるかどうかを検出するのに役立ちます。8500 バイトを超えるパケットは VPC エンドポイントに到達したときにドロップされることに注意してください。トラブルシューティングについては、「インターフェイス VPC エンドポイントとエンドポイントサービス間の接続の問題」を参照してください。

目的: このアラームは、エンドポイントやエンドポイントサービスに異常があるかどうかを検出するのに使用されます。

統計: Sum

推奨しきい値: 状況によって異なります。

しきい値の根拠: ユースケースに応じてしきい値を設定します。エンドポイントまたはエンドポイントサービスの異常状態を把握したい場合は、データが大幅に失われる前に問題を解決できるよう、しきい値を低く設定する必要があります。履歴データを使用して、ドロップされるパケット数の許容範囲を把握し、それに応じてしきい値を設定できます。

期間: 60

アラームを発生させるデータポイント数: 5

評価期間数: 5

比較演算子: GREATER_THAN_THRESHOLD

AWS プライベートリンク (**AWS/PrivateLinkServices**)

RstPacketsSent

ディメンション: Service Id、Load Balancer Arn、Az

アラームの説明: このアラームは、エンドポイントに送信されたリセットパケットの数に基づいて、エンドポイントサービスの異常なターゲットを検出するのに役立ちます。サービスのコンシューマーとの接続エラーをデバッグする際に、サービスが RstPacketsSent メトリクスによって接続をリセットしているのか、それともネットワークパス上で何か他の障害が発生しているのかを検証できます。

目的: このアラームは、エンドポイントサービスの異常なターゲットを検出するのに使用されます。

統計: Sum

推奨しきい値: 状況によって異なります。

しきい値の根拠: しきい値はユースケースによって異なります。ユースケースでターゲットの異常を許容できる場合は、しきい値を高く設定できます。ユースケースで異常なターゲットを許容できない場合は、しきい値を非常に低く設定できます。

期間: 60

アラームを発生させるデータポイント数: 5

評価期間数: 5

比較演算子: GREATER_THAN_THRESHOLD

Amazon RDS

CPUUtilization

ディメンション: DBInstanceIdentifier

アラームの説明: このアラームは、常に高い CPU 使用率をモニタリングするのに役立ちます。CPU 使用率は、アイドル状態でない時間を測定します。拡張モニタリングまたは Performance Insights を使用して、MariaDB、MySQL、Oracle、および PostgreSQL の CPU 時間 (guest、irq、wait、nice など) を最も多く消費している待機時間を確認することを検討してください。次に、CPU を最も多く消費しているクエリを評価します。ワークロードを調整できない場合は、より大きな DB インスタンスクラスへの移行を検討してください。

目的: このアラームは、常に高い CPU 使用率を検出して、非常に長い応答時間やタイムアウトを防ぐために使用されます。CPU 使用率がマイクロバーストしていることを確認したい場合は、アラームの評価時間を短く設定できます。

統計: Average

推奨しきい値: 90.0

しきい値の根拠: CPU 使用率がランダムに増加してもデータベースのパフォーマンスは低下しないかもしれませんが、CPU 使用率が高い状態が続くと、今後のデータベースリクエストが妨げられる可能性があります。データベース全体のワークロードによっては、RDS/Aurora インスタンスの CPU 使用率が高いと、全体的なパフォーマンスが低下する可能性があります。

期間: 60

アラームを発生させるデータポイント数: 5

評価期間数: 5

比較演算子: GREATER_THAN_THRESHOLD

DatabaseConnections

ディメンション: DBInstanceIdentifier

アラームの説明: このアラームは多数の接続を検出します。既存の接続を確認し、「スリープ」状態の接続や不適切に閉じられた接続を終了します。接続プールを使用して新しい接続の数を制限することを検討してください。または、DB インスタンスのサイズを増やしてメモリの多いクラスを使用するため、「max_connections」のデフォルト値を高くするか、現在のクラスがワークロードをサポートできる場合は、現在のクラスの RDS、Aurora MySQL、および PostgreSQL で「max_connections」値を増やします。

目的: このアラームは、DB 接続の最大数に達したときに接続が拒否されないようにするために使用されます。DB インスタンスクラスを頻繁に変更する場合は、メモリとデフォルトの最大接続数が変わるため、このアラームはお勧めしません。

統計: Average

推奨しきい値: 状況によって異なります。

しきい値の根拠: 許可される接続数は、DB インスタンスクラスのサイズと、プロセス/接続に関連するデータベースエンジン固有のパラメータによって異なります。データベースの最大接続数の 90 ~ 95% の値を計算し、その結果をしきい値として使用する必要があります。

期間: 60

アラームを発生させるデータポイント数: 5

評価期間数: 5

比較演算子: GREATER_THAN_THRESHOLD

EBSByteBalance%

ディメンション: DBInstanceIdentifier

アラームの説明: このアラームは、残っているスループットクレジットの割合が低いことを監視するのに役立ちます。トラブルシューティングについては、「RDS のレイテンシー問題」を参照してください。

目的: このアラームは、バーストバケットに残っているスループットクレジットの割合が低いことを検出するために使用されます。バイトバランスの割合が低いと、スループットボトルネックの問題が発生する可能性があります。このアラームは、Aurora PostgreSQL インスタンスにはお勧めしません。

統計: Average

推奨しきい値: 10.0

しきい値の根拠: スループットクレジット残高が 10% を下回ると不十分と見なされるため、状況に応じてしきい値を設定する必要があります。アプリケーションがワークロードのスループットの低下を許容できる場合は、より低いしきい値を設定することもできます。

期間: 60

アラームを発生させるデータポイント数: 3

評価期間数: 3

比較演算子: LESS_THAN_THRESHOLD

EBSIOBalance%

ディメンション: DBInstanceIdentifier

アラームの説明: このアラームは、残っている IOPS クレジットの割合が低いことを監視するのに役立ちます。レイテンシー問題のトラブルシューティングについては、「RDS のレイテンシー問題」を参照してください。

目的: このアラームは、バーストバケットに残っている I/O クレジットの割合が低いことを検出するために使用されます。IOPS バランスの割合が低いと、IOPS ボトルネックの問題が発生する可能性があります。このアラームは、Aurora インスタンスにはお勧めしません。

統計: Average

推奨しきい値: 10.0

しきい値の根拠: IOPS クレジット残高が 10% を下回ると不十分と見なされるため、状況に応じてしきい値を設定できます。アプリケーションがワークロードの IOPS の低下を許容できる場合は、より低いしきい値を設定することもできます。

期間: 60

アラームを発生させるデータポイント数: 3

評価期間数: 3

比較演算子: LESS_THAN_THRESHOLD

FreeableMemory

ディメンション: DBInstanceIdentifier

アラームの説明: このアラームは、データベース接続が急増している場合や、インスタンスのメモリプレッシャーが高くなっている場合など、解放可能なメモリが不足していることを監視するのに役立ちます。`FreeableMemory` に加えて `SwapUsage` の CloudWatch メトリクスを監視して、メモリプレッシャーを確認します。インスタンスメモリの消費量が頻繁に高くなりすぎている場合は、ワークロードの見直し、またはインスタンスクラスのアップグレードが必要であることを表します。Aurora リーダー DB インスタンスの場合は、クラスターにリーダー DB インスタンスを追加することを検討してください。Aurora のトラブルシューティングの詳細については、「解放可能なメモリの問題」を参照してください。

目的: このアラームは、メモリが不足して接続が拒否されるのを防ぐのに役立ちます。

統計: Average

推奨しきい値: 状況によって異なります。

しきい値の根拠: ワークロードとインスタンスクラスによっては、適切なしきい値が異なる場合があります。理想的には、使用可能なメモリが長時間にわたって合計メモリの 25% を下回らないようにする必要があります。Aurora の場合、メトリクスが 0 に近づくと DB インスタンスが可能な限りスケールアップしたことを意味するため、しきい値を 5% 近くに設定できます。このメトリクスの過去の動作を分析して、適切なしきい値レベルを決定できます。

期間: 60

アラームを発生させるデータポイント数: 15

評価期間数: 15

比較演算子: LESS_THAN_THRESHOLD

FreeLocalStorage

ディメンション: DBInstanceIdentifier

アラームの説明: このアラームは、空き容量が少ないローカルストレージを監視するのに役立ちます。Aurora PostgreSQL 互換エディションは、ローカルストレージを使用してエラーログと一時ファイルを保存します。Aurora MySQL は、エラーログ、全般ログ、スロークエリログ、監査ログ、および InnoDB 以外の一時テーブルを保存するのにローカルストレージを使用します。これらのローカルストレージボリュームは、Amazon EBS Store によってバックアップされ、より大きな DB インスタンスクラスを使用することで拡張できます。トラブルシューティングについては、Aurora「PostgreSQL 互換」と「MySQL 互換」を参照してください。

目的: このアラームは、Aurora Serverless v2 以降を使用していない場合に、Aurora DB インスタンスがローカルストレージの制限にどれだけ近づいているかを検出するために使用されます。一時テーブルやログファイルなどの非永続的データをローカルストレージに保存すると、ローカルストレージが容量に達する可能性があります。このアラームは、DB インスタンスがローカルストレージを使い果たしたときに発生する容量不足エラーを防ぐことができます。

統計: Average

推奨しきい値: 状況によって異なります。

しきい値の根拠: ボリュームの使用速度と傾向に基づいて使用可能なストレージ量の約 10% ~ 20% を計算し、その結果をしきい値として使用して、ボリュームが制限に達する前に事前に措置を講じる必要があります。

期間: 60

アラームを発生させるデータポイント数: 5

評価期間数: 5

比較演算子: LESS_THAN_THRESHOLD

FreeStorageSpace

ディメンション: DBInstanceIdentifier

アラームの説明: このアラームは、使用可能なストレージ容量が少ないことを監視します。ストレージ容量の制限に頻繁に近づく場合は、データベースストレージをスケールアップすることを検討してください。アプリケーションからのリクエストの予期しない増加に対応するために、いくらかのバッファを含めてください。または、RDS ストレージ自動スケーリングを有効にすることを検討してください。さらに、未使用または古いデータや口グを削除して、空き容量を増やすことを検討してください。詳細については、「RDS ストレージ不足に関するドキュメント」と「PostgreSQL ストレージ問題に関するドキュメント」を参照してください。

目的: このアラームは、ストレージがいっぱいになる問題を防ぐのに役立ちます。これにより、データベースインスタンスがストレージを使い果たしたときに発生するダウンタイムを防ぐことができます。ストレージ自動スケーリングが有効になっている場合や、データベースインスタンスのストレージ容量を頻繁に変更する場合は、このアラームを使用することはお勧めしません。

統計: Minimum

推奨しきい値: 状況によって異なります。

しきい値の根拠: しきい値は、現在割り当てられているストレージ容量によって異なります。通常は、割り当てられているストレージ容量の 10 パーセントの値を計算し、その結果をしきい値として使用する必要があります。

期間: 60

アラームを発生させるデータポイント数: 5

評価期間数: 5

比較演算子: LESS_THAN_THRESHOLD

MaximumUsedTransactionIDs

ディメンション: DBInstanceIdentifier

アラームの説明: このアラームは、PostgreSQL のトランザクション ID のラップアラウンドを防ぐのに役立ちます。この問題を調査して解決するには、このブログのトラブルシューティング手順を参照してください。このブログを参照して、autovacuum の概念、一般的な問題、ベストプラクティスについてさらに理解を深めることもできます。

目的: このアラームは、PostgreSQL のトランザクション ID のラップアラウンドを防ぐのに役立ちます。

統計: Average

推奨しきい値: 1.0E9

しきい値の根拠: このしきい値を 10 億に設定すると、問題を調査する時間を確保できます。autovacuum_freeze_max_age のデフォルト値は 2 億です。最も古いトランザクションの存続時間が 10 億の場合、autovacuum がこのしきい値を目標の 2 億トランザクション ID 以下に維持することに問題があります。

期間: 60

アラームを発生させるデータポイント数: 1

評価期間数: 1

比較演算子: GREATER_THAN_THRESHOLD

ReadLatency

ディメンション: DBInstanceIdentifier

アラームの説明: このアラームは、大きな読み取りレイテンシーを監視するのに役立ちます。ストレージレイテンシーが大きい場合は、ワークロードがリソースの制限を超えていることが原因です。インスタンスと割り当てられたストレージ構成を基準にして I/O 使用率を確認できます。「IOPS ボトルネックによる Amazon EBS ボリュームのレイテンシーのトラブルシューティング」を参照してください。Aurora では、I/O 最適化ストレージ構成のインスタンスクラスに切り替えることができます。ガイダンスについては、「Planning I/O in Aurora」を参照してください。

目的: このアラームは、大きい読み取りレイテンシーを検出するために使用されます。データベースディスクは通常、読み取り/書き込みレイテンシーは小さいですが、レイテンシーが大きいオペレーションを引き起こすような問題が発生する可能性があります。

統計: p90

推奨しきい値: 状況によって異なります。

しきい値の根拠: このアラームの推奨しきい値は、ユースケースによって大きく異なります。20 ミリ秒を超える読み取りレイテンシーは、調査の対象となる可能性があります。また、アプリケーションの読み取りオペレーションのレイテンシーが大きくなる可能性がある場合は、しきい値を高く設定することもできます。読み取りレイテンシーの重要度と要件を確認し、このメトリクスの過去の動作を分析して、適切なしきい値レベルを決定します。

期間: 60

アラームを発生させるデータポイント数: 5

評価期間数: 5

比較演算子: GREATER_THAN_THRESHOLD

ReplicaLag

ディメンション: DBInstanceIdentifier

アラームの説明: このアラームは、レプリカがプライマリインスタンスより遅れている秒数を把握するのに役立ちます。ソースデータベースインスタンスでユーザートランザクションが発生していない場合、PostgreSQL リードレプリカで最長 5 分のレプリケーションの遅延が報告されます。ReplicaLag メトリクスが 0 に達すると、レプリカはプライマリ DB インスタンスに追いついています。ReplicaLag メトリクスにより -1 が返された場合、レプリケーションは現在アクティブではありません。RDS PostgreSQL に関するガイダンスについては、「レプリケーションのベストプラクティス」を、ReplicaLag のトラブルシューティングと関連するエラーについては、「ReplicaLag のトラブルシューティング」を参照してください。

目的: このアラームは、プライマリインスタンスに障害が発生した場合に生じる可能性のあるデータ損失を反映するレプリカラグを検出できます。レプリカがプライマリよりも大幅に遅れ、プライマリに障害が発生した場合、レプリカでは、プライマリインスタンスにあったデータが失われます。

統計: Maximum

推奨しきい値: 60.0

しきい値の根拠: 通常、許容できる遅延はアプリケーションによって異なります。60 秒を超えないことをお勧めします。

期間: 60

アラームを発生させるデータポイント数: 10

評価期間数: 10

比較演算子: GREATER_THAN_THRESHOLD

WriteLatency

ディメンション: DBInstanceIdentifier

アラームの説明: このアラームは、大きな書き込みレイテンシーを監視するのに役立ちます。ストレージレイテンシーが大きい場合は、ワークロードがリソースの制限を超えていることが原因です。インスタンスと割り当てられたストレージ構成を基準にして I/O 使用率を確認できます。「IOPS ボトルネックによる Amazon EBS ボリュームのレイテンシーのトラブルシューティング」を参照してください。Aurora では、I/O 最適化ストレージ構成のインスタンスクラスに切り替えることができます。ガイダンスについては、「Planning I/O in Aurora」を参照してください。

目的: このアラームは、大きい書き込みレイテンシーを検出するために使用されます。データベースディスクは通常、読み取り/書き込みレイテンシーは小さいですが、レイテンシーが大きいオペレーションを引き起こすような問題が発生する可能性があります。これを監視することで、ディスクレイテンシーが予想どおりに小さくなることが保証されます。

統計: p90

推奨しきい値: 状況によって異なります。

しきい値の根拠: このアラームの推奨しきい値は、ユースケースによって大きく異なります。20 ミリ秒を超える書き込みレイテンシーは、調査の対象となる可能性があります。また、アプリケーションの書き込みオペレーションのレイテンシーが大きくなる可能性がある場合は、しきい値を高く設定することもできます。書き込みレイテンシーの重要度と要件を確認し、このメトリクスの過去の動作を分析して、適切なしきい値レベルを決定します。

期間: 60

アラームを発生させるデータポイント数: 5

評価期間数: 5

比較演算子: GREATER_THAN_THRESHOLD

DBLoad

ディメンション: DBInstanceIdentifier

アラームの説明: このアラームは、高い DB ロードを監視するのに役立ちます。プロセスの数が vCPU の数を超えると、プロセスはキューイングを開始します。キューイングが増加すると、パフォーマンスに影響します。DB ロードが最大 vCPU ラインをしばしば超過し、プライマリ待機状態が CPU である場合、CPU が過ロードになっています。この場合、`CPUUtilization`、`DBLoadCPU`、およびキューに入っているタスクを Performance Insights/拡張モニタリングで監視することができます。インスタンスへの接続を抑制したり、CPU ロードの高い SQL クエリを調整したり、より大きなインスタンスクラスを検討する必要があります。待機状態の高い一貫したインスタンスは、解決するボトルネックまたはリソースの競合問題がある可能性があることを示します。

目的: このアラームは、DB ロードが高いことを検出するために使用されます。DB ロードが高いと、DB インスタンスにパフォーマンスの問題が発生する可能性があります。このアラームは、サーバーレス DB インスタンスには適用されません。

統計: Average

推奨しきい値: 状況によって異なります。

しきい値の根拠: 最大 vCPU 値は、DB インスタンスの vCPU (仮想 CPU) のコア数によって決まります。最大 vCPU によって、適切なしきい値が異なる場合があります。理想的には、DB ロードは vCPU ラインを超えないようにしてください。

期間: 60

アラームを発生させるデータポイント数: 15

評価期間数: 15

比較演算子: GREATER_THAN_THRESHOLD

AuroraVolumeBytesLeftTotal

ディメンション: DBClusterIdentifier

アラームの説明: このアラームは、残りの合計ボリュームが少ないことを監視するのに役立ちます。残りの合計ボリュームがサイズ制限に達すると、クラスターはスペースがないというエラー

を報告します。Aurora ストレージはクラスターボリューム内のデータに合わせて自動的にスケールし、DB エンジンのバージョンに応じて最大 128 TiB または 64 TiB まで拡張されます。不要になったテーブルとデータベースを削除して、ストレージを削減することを検討してください。詳細については、「ストレージのスケーリング」を参照してください。

目的: このアラームは、Aurora クラスターがボリュームサイズの制限にどれだけ近づいているかを検出するために使用されます。このアラームは、クラスターの容量が不足したときに発生する容量不足エラーを防ぐことができます。このアラームは、Aurora MySQL にのみお勧めします。

統計: Average

推奨しきい値: 状況によって異なります。

しきい値の根拠: ボリューム使用量の増加速度と傾向に基づいて実際のサイズ制限の 10% ~ 20% を計算し、その結果をしきい値として使用して、ボリュームが制限に達する前に事前に措置を講じる必要があります。

期間: 60

アラームを発生させるデータポイント数: 5

評価期間数: 5

比較演算子: LESS_THAN_THRESHOLD

AuroraBinlogReplicaLag

ディメンション: DBClusterIdentifier、Role=WRITER

アラームの説明: このアラームは、Aurora ライターインスタンスのレプリケーションのエラー状態を監視するのに役立ちます。詳細については、「AWS リージョン間での Aurora MySQL DB クラスターのレプリケート」を参照してください。トラブルシューティングについては、「Aurora MySQL レプリケーションの問題」を参照してください。

目的: このアラームは、ライターインスタンスがエラー状態にあり、ソースを複製できないかどうかを検出するために使用されます。このアラームは、Aurora MySQL にのみお勧めします。

統計: Average

推奨しきい値: -1.0

しきい値の根拠: レプリカがエラー状態にある場合、Aurora MySQL はこの値を公開するため、しきい値として -1 を使用することをお勧めします。

期間: 60

アラームを発生させるデータポイント数: 2

評価期間数: 2

比較演算子: LESS_THAN_OR_EQUAL_TO_THRESHOLD

BlockedTransactions

ディメンション: DBInstanceIdentifier

アラームの説明: このアラームは、Aurora DB インスタンスでブロックされたトランザクション数が多いことを監視するのに役立ちます。ブロックされたトランザクションは、ロールバックまたはコミットのいずれかで終了することがあります。高い同時実行性、トランザクションのアイドル状態、または実行時間の長いトランザクションにより、トランザクションがブロックされる可能性があります。トラブルシューティングについては、「Aurora MySQL」に関するドキュメントを参照してください。

目的: このアラームは、トランザクションのロールバックやパフォーマンスの低下を防ぐために、Aurora DB インスタンスでブロックされたトランザクションの数が多いことを検出するために使用されます。

統計: Average

推奨しきい値: 状況によって異なります。

しきい値の根拠: `ActiveTransactions` メトリクスを使用してインスタンスの全トランザクションの 5% を計算し、その結果をしきい値として使用する必要があります。また、ブロックされたトランザクションの重要度と要件を確認し、このメトリクスの過去の動作を分析して、適切なしきい値を決定することもできます。

期間: 60

アラームを発生させるデータポイント数: 5

評価期間数: 5

比較演算子: GREATER_THAN_THRESHOLD

BufferCacheHitRatio

ディメンション: DBInstanceIdentifier

アラームの説明: このアラームは、Aurora クラスターのキャッシュヒット率が常に低いことを監視するのに役立ちます。ヒット率が低い場合、この DB インスタンスのクエリが、頻繁にディスクに書き込まれていることを示します。トラブルシューティングについては、ワークロードを調査して、この動作の原因となっているクエリを確認し、「DB インスタンスの RAM の推奨事項」に関するドキュメントを参照してください。

目的: このアラームは、Aurora インスタンスのパフォーマンスが持続的に低下するのを防ぐために、キャッシュヒット率が常に低いことを検出するために使用されます。

統計: Average

推奨しきい値: 80.0

しきい値の根拠: バッファキャッシュヒット率のしきい値を 80% に設定できます。ただし、許容できるパフォーマンスレベルとワークロード特性に基づいてこの値を調整できます。

期間: 60

アラームを発生させるデータポイント数: 10

評価期間数: 10

比較演算子: LESS_THAN_THRESHOLD

EngineUptime

ディメンション: DBClusterIdentifier、Role=WRITER

アラームの説明: このアラームは、ライター DB インスタンスのダウンタイムが短いことを監視するのに役立ちます。ライター DB インスタンスは、再起動、メンテナンス、アップグレード、またはフェイルオーバーによりダウンする可能性があります。クラスター内のフェイルオーバーによってアップタイムが 0 になり、クラスターに 1 つ以上の Aurora レプリカがある場合は、障害イベント中に 1 つの Aurora レプリカがプライマリライターインスタンスに昇格されます。DB クラスターの可用性を高めるために、複数のアベイラビリティーゾーン内で 1 つ以上の Aurora レプリカを作成することを検討してください。詳細については、「Aurora のダウンタイムに影響する要因」を参照してください。

目的: このアラームは、Aurora ライター DB インスタンスがダウンタイム状態にあるかどうかを検出するために使用されます。これにより、クラッシュやフェイルオーバーが原因でライターインスタンスに長時間障害が発生するのを防ぐことができます。

統計: Average

推奨しきい値: 0.0

しきい値の根拠: 障害イベントによって短い中断が発生し、その間例外によって読み取りと書き込みオペレーションが失敗します。ただし、サービスの一般的な復元時間は 60 秒未満であり、多くの場合 30 秒未満です。

期間: 60

アラームを発生させるデータポイント数: 2

評価期間数: 2

比較演算子: LESS_THAN_OR_EQUAL_TO_THRESHOLD

RollbackSegmentHistoryListLength

ディメンション: DBInstanceIdentifier

アラームの説明: このアラームは、Aurora インスタンスのロールバックセグメント履歴の長さが一貫して長くなっていることを監視するのに役立ちます。InnoDB 履歴リストの長さが大きくなると、古い行バージョンが多数存在することになり、クエリやデータベースのシャットダウンが遅くなります。詳細とトラブルシューティングについては、「InnoDB 履歴リストの長さが大幅に増加した」に関するドキュメントを参照してください。

目的: このアラームは、ロールバックセグメント履歴の長さが一貫して長くなっていることを検出するために使用されます。これにより、Aurora インスタンスでパフォーマンスが継続的に低下したり、CPU 使用率が高くなったりするのを防ぐことができます。このアラームは、Aurora MySQL にのみお勧めします。

統計: Average

推奨しきい値: 1000000.0

しきい値の根拠: このしきい値を 100 万に設定すると、問題を調査する時間を確保できます。ただし、許容できるパフォーマンスレベルとワークロード特性に基づいてこの値を調整できます。

期間: 60

アラームを発生させるデータポイント数: 5

評価期間数: 5

比較演算子: GREATER_THAN_THRESHOLD

StorageNetworkThroughput

ディメンション: DBClusterIdentifier、Role=WRITER

アラームの説明: このアラームは、高いストレージネットワークスループットを監視するのに役立ちます。ストレージネットワークスループットが EC2 インスタンスのネットワーク帯域幅の合計を超えると、読み取りと書き込みのレイテンシーが高くなり、パフォーマンスが低下する可能性があります。EC2 インスタンスタイプは、AWS コンソールで確認できます。トラブルシューティングについては、書き込み/読み取りレイテンシーの変化を確認し、このメトリクスでアラームが発生しているかどうかも評価してください。アラームが発生している場合は、アラームがトリガーされた時間帯のワークロードパターンを評価してください。これにより、ワークロードを最適化してネットワークトラフィックの総量を削減できるかどうかを判断できます。これが不可能な場合は、インスタンスのスケーリングを検討する必要があるかもしれません。

目的: このアラームは、ストレージネットワークスループットが高いことを検出するために使用されます。高スループットを検出することで、ネットワークパケットのドロップやパフォーマンスの低下を防ぐことができます。

統計: Average

推奨しきい値: 状況によって異なります。

しきい値の根拠:EC2 インスタンスタイプの総ネットワーク帯域幅の 80% ~ 90% を計算し、その結果を閾値として使用して、ネットワークパケットが影響を受ける前に事前に措置を講じる必要があります。また、ストレージネットワークスループットの重要度と要件を確認し、このメトリクスの過去の動作を分析して、適切なしきい値を決定することもできます。

期間: 60

アラームを発生させるデータポイント数: 5

評価期間数: 5

比較演算子: GREATER_THAN_THRESHOLD

Amazon Route 53 Public Data Plane

HealthCheckStatus

ディメンション: HealthCheckId

アラームの説明: このアラームは、ヘルスチェッカーに従って異常のあるエンドポイントを検出するのに役立ちます。異常なステータスになる障害の原因を理解するには、Route 53 ヘルスチェックコンソールの [ヘルスチェッカー] タブを使用して、各リージョンのステータスと、ヘルスチェックで検出した最後の障害を表示します。[ステータス] タブには、エンドポイントが異常と報告された理由も表示されます。トラブルシューティングの手順を参照してください。

目的: このアラームは Route53 ヘルスチェッカーを使用して異常のあるエンドポイントを検出します。

統計: Average

推奨しきい値: 1.0

しきい値の根拠: エンドポイントのステータスは、正常であれば 1 と報告されます。1 未満はすべて異常です。

期間: 60

アラームを発生させるデータポイント数: 3

評価期間数: 3

比較演算子: LESS_THAN_THRESHOLD

Amazon S3

4xxErrors

ディメンション: BucketName、FilterId

アラームの説明: このアラームは、クライアントのリクエストに応答して生成されたエラーステータスコード 4xx の総数を報告するのに役立ちます。例えば、エラーコード 403 は IAM ポリシーが正しくないことを示している可能性があり、エラーコード 404 はクライアントアプリケーションの動作が正しくないことを示している可能性があります。一時的に S3 サーバーアクセスログを有効にすると、HTTP ステータスフィールドと Error Code フィールドを使用して問題の原因を特定するのに役立ちます。エラーコードの詳細については、「エラーレスポンス」を参照してください。

目的: このアラームは、通常の 4xx エラー率のベースラインを作成するのに使用されます。これにより、セットアップの問題を示す可能性のある異常を調べることができます。

統計: Average

推奨しきい値: 0.05

しきい値の根拠: 推奨しきい値は、全リクエストの 5% 超が 4XX エラーになっていることを検出するためのものです。頻繁に発生する 4XX エラーにはアラームが必要です。ただし、しきい値が非常に低いと、アラームの感度が高くなりすぎる可能性があります。また、4XX エラーの許容レベルを考慮して、リクエストの負荷に合わせてしきい値を調整することもできます。また、履歴データを分析してアプリケーションワークロードの許容可能なエラー率を確認し、それに応じてしきい値を調整することもできます。

期間: 60

アラームを発生させるデータポイント数: 15

評価期間数: 15

比較演算子: GREATER_THAN_THRESHOLD

5xxErrors

ディメンション: BucketName、FilterId

アラームの説明: このアラームは、サーバー側エラー数が多いことを検出するのに役立ちます。これらのエラーは、クライアントがリクエストを行ったが、サーバーが完了できなかったことを示しています。これは、S3 が原因でアプリケーションが直面している問題を関連付けることに役立ちます。エラーを効率的に処理または削減するのに役立つ詳細については、「パフォーマンスを最適化する設計パターン」を参照してください。エラーは S3 の問題が原因である可能性もあります。AWS サービスヘルスダッシュボードで、リージョンの Amazon S3 のステータスを確認してください。

目的: このアラームは、5xx エラーが原因でアプリケーションに問題が発生しているかどうかを検出するのに役立ちます。

統計: Average

推奨しきい値: 0.05

しきい値の根拠: 全リクエストの 5% 超が 5XXError になっている場合に検出されるように、しきい値を設定することをお勧めします。ただし、リクエストのトラフィックや許容可能なエラー率に合わせてしきい値を調整できます。また、履歴データを分析してアプリケーションワークロードの許容可能なエラー率を確認し、それに応じてしきい値を調整することもできます。

期間: 60

アラームを発生させるデータポイント数: 15

評価期間数: 15

比較演算子: GREATER_THAN_THRESHOLD

OperationsFailedReplication

ディメンション: SourceBucket、DestinationBucket、RuleId

アラームの説明: このアラームは、レプリケーションの障害を把握するのに役立ちます。このメトリックスは、S3 CRR または S3 SRR を使用してレプリケートされた新しいオブジェクトのステータスを追跡し、S3 バッチレプリケーションを使用してレプリケートされた既存のオブジェクトも追跡します。詳細については、「レプリケーションのトラブルシューティング」を参照してください。

目的: このアラームは、失敗したレプリケーションオペレーションがあるかどうかを検出するために使用されます。

統計: Maximum

推奨しきい値: 0.0

しきい値の根拠: このメトリクスは、オペレーションが成功した場合は 0 の値を出力し、1 分間レプリケーションオペレーションが実行されなかった場合は何も出力しません。メトリクスが 0 より大きい値を出力した場合は、レプリケーションオペレーションが失敗しています。

期間: 60

アラームを発生させるデータポイント数: 5

評価期間数: 5

比較演算子: GREATER_THAN_THRESHOLD

S3ObjectLambda

4xxErrors

ディメンション: AccessPointName、DataSourceARN

アラームの説明: このアラームは、クライアントのリクエストに応答して生成されたエラーステータスコード 4xx の総数を報告するのに役立ちます。一時的に S3 サーバーアクセスログを有効にすると、HTTP ステータスフィールドと Error Code フィールドを使用して問題の原因を特定するのに役立ちます。

目的: このアラームは、通常の 4xx エラー率のベースラインを作成するのに使用されます。これにより、セットアップの問題を示す可能性のある異常を調べることができます。

統計: Average

推奨しきい値: 0.05

しきい値の根拠: 全リクエストの 5% 超が 4XXError になっている場合に検出されるように、しきい値を設定することをお勧めします。頻繁に発生する 4XX エラーにはアラームが必要です。ただし、しきい値が非常に低いと、アラームの感度が高くなりすぎる可能性があります。また、4XX エラーの許容レベルを考慮して、リクエストの負荷に合わせてしきい値を調整することもできます。また、履歴データを分析してアプリケーションワークロードの許容可能なエラー率を確認し、それに応じてしきい値を調整することもできます。

期間: 60

アラームを発生させるデータポイント数: 15

評価期間数: 15

比較演算子: GREATER_THAN_THRESHOLD

5xxErrors

ディメンション: AccessPointName、DataSourceARN

アラームの説明: このアラームは、サーバー側エラー数が多いことを検出するのに役立ちます。これらのエラーは、クライアントがリクエストを行ったが、サーバーが完了できなかったことを示しています。これらのエラーは S3 の問題が原因である可能性があります。AWS サービスヘルスダッシュボードで、リージョンの Amazon S3 のステータスを確認してください。これは、S3 が原因でアプリケーションが直面している問題を関連付けることに役立ちます。これらのエラーを効率的に処理または軽減するのに役立つ情報については、「パフォーマンスを最適化する設計パターン」を参照してください。

目的: このアラームは、5xx エラーが原因でアプリケーションに問題が発生しているかどうかを検出するのに役立ちます。

統計: Average

推奨しきい値: 0.05

しきい値の根拠: 全リクエストの 5% 超が 5XX エラーになっている場合に検出されるように、しきい値を設定することをお勧めします。ただし、リクエストのトラフィックや許容可能なエラー率に合わせてしきい値を調整できます。また、履歴データを分析してアプリケーションワークロードの許容可能なエラー率を確認し、それに応じてしきい値を調整することもできます。

期間: 60

アラームを発生させるデータポイント数: 15

評価期間数: 15

比較演算子: GREATER_THAN_THRESHOLD

LambdaResponse4xx

ディメンション: AccessPointName、DataSourceARN

アラームの説明: このアラームは、S3 Object Lambda への呼び出しの失敗 (500 番台) を検出して診断するのに役立ちます。これらのエラーは、リクエストへのレスポンスを担当する Lambda 関数のエラーまたは設定ミスが原因である可能性があります。Object Lambda アクセスポイントに関連付けられた Lambda 関数の CloudWatch ログストリームを調査すると、S3 Object Lambda からのレスポンスに基づいて問題の原因を特定するのに役立ちます。

目的: このアラームは、WriteGetObjectResponse 呼び出しのクライアントエラー 4xx を検出するために使用されます。

統計: Average

推奨しきい値: 0.05

しきい値の根拠: 全リクエストの 5% 超が 4XXError になっている場合に検出されるように、しきい値を設定することをお勧めします。頻繁に発生する 4XX エラーにはアラームが必要です。ただし、しきい値が非常に低いと、アラームの感度が高くなりすぎる可能性があります。また、4XX エラーの許容レベルを考慮して、リクエストの負荷に合わせてしきい値を調整することもできます。また、履歴データを分析してアプリケーションワークロードの許容可能なエラー率を確認し、それに応じてしきい値を調整することもできます。

期間: 60

アラームを発生させるデータポイント数: 15

評価期間数: 15

比較演算子: GREATER_THAN_THRESHOLD

Amazon SNS

NumberOfMessagesPublished

ディメンション: TopicName

アラームの説明: このアラームは、公開されている SNS メッセージの数が少なすぎることを検出できます。トラブルシューティングのため、パブリッシャーが送信するトラフィックが減少している理由を確認します。

目的: このアラームは、通知の公開が大幅に低下していることを事前にモニタリングして検出するのに役立ちます。これにより、アプリケーションまたはビジネスプロセスで発生する可能性のある問題を特定できるため、通知のフローを想定どおりに維持するための適切なアクションを実行できます。システムが処理するトラフィックが最小限になると予想される場合は、このアラームを作成する必要があります。

統計: Sum

推奨しきい値: 状況によって異なります。

しきい値の根拠: 公開されるメッセージの数は、アプリケーションで公開されるメッセージの予想数と一致している必要があります。また、履歴データ、傾向、トラフィックを分析して、適切なしきい値を見つけることもできます。

期間: 60

アラームを発生させるデータポイント数: 5

評価期間数: 5

比較演算子: LESS_THAN_THRESHOLD

NumberOfNotificationsDelivered

ディメンション: TopicName

アラームの説明: このアラームは、配信された SNS メッセージの数が少なすぎることを検出できます。これは、エンドポイントのサブスクリプションが意図せず解除されたことや、SNS イベントが原因でメッセージに遅延が発生したことが原因である可能性があります。

目的: このアラームは、配信されるメッセージ量の減少を検出するのに役立ちます。システムが処理するトラフィックが最小限になると予想される場合は、このアラームを作成する必要があります。

統計: Sum

推奨しきい値: 状況によって異なります。

しきい値の根拠: 配信されるメッセージの数は、メッセージの予想数およびコンシューマーの数と一致している必要があります。また、履歴データ、傾向、トラフィックを分析して、適切なしきい値を見つけることもできます。

期間: 60

アラームを発生させるデータポイント数: 5

評価期間数: 5

比較演算子: LESS_THAN_THRESHOLD

NumberOfNotificationsFailed

ディメンション: TopicName

アラームの説明: このアラームは、失敗した SNS メッセージの数が多すぎることを検出できます。通知の失敗をトラブルシューティングするには、CloudWatch Logs のログ記録を有効にします。ログを確認すると、失敗しているサブスクライバーと、それらのサブスクライバーが返しているステータスコードを確認するのに役立ちます。

目的: このアラームは、通知の配信に関する問題を事前に発見し、それに対処するための適切なアクションを実行するのに役立ちます。

統計: Sum

推奨しきい値: 状況によって異なります。

しきい値の根拠: このアラームの推奨しきい値は、通知の失敗の影響度によって大きく異なります。エンドユーザーに提供される SLA、耐障害性、通知の重要性を確認し、履歴データを分析

し、それに応じてしきい値を選択します。SQS、Lambda、または Firehose サブスクリプションしかないトピックでは、失敗した通知の数は 0 になるはずです。

期間: 60

アラームを発生させるデータポイント数: 5

評価期間数: 5

比較演算子: GREATER_THAN_THRESHOLD

NumberOfNotificationsFilteredOut-InvalidAttributes

ディメンション: TopicName

アラームの説明: このアラームは、パブリッシャーまたはサブスクライバーに発生する可能性のある問題をモニタリングして解決するのに役立ちます。パブリッシャーが無効な属性を持つメッセージを公開していないか、サブスクライバーに不適切なフィルターが適用されていないかを確認します。また、CloudWatch Logs を分析して、問題の根本的な原因の発見に役立てることもできます。

目的: このアラームは、公開されたメッセージが有効でないか、サブスクライバーに不適切なフィルターが適用されているかを検出するために使用されます。

統計: Sum

推奨しきい値: 0.0

しきい値の根拠: 無効な属性は、ほとんどの場合、パブリッシャーのミスです。正常なシステムでは無効な属性は想定されないため、しきい値を 0 に設定することをおすすめします。

期間: 60

アラームを発生させるデータポイント数: 5

評価期間数: 5

比較演算子: GREATER_THAN_THRESHOLD

NumberOfNotificationsFilteredOut-InvalidMessageBody

ディメンション: TopicName

アラームの説明: このアラームは、パブリッシャーまたはサブスクライバーに発生する可能性のある問題をモニタリングして解決するのに役立ちます。パブリッシャーが無効なメッセージ本文

を含むメッセージを公開していないか、またはサブスクライバーに不適切なフィルターが適用されていないかを確認します。また、CloudWatch Logs を分析して、問題の根本的な原因の発見に役立てることもできます。

目的: このアラームは、公開されたメッセージが有効でないか、サブスクライバーに不適切なフィルターが適用されているかを検出するために使用されます。

統計: Sum

推奨しきい値: 0.0

しきい値の根拠: 無効なメッセージ本文は、ほとんどの場合、パブリッシャーのミスです。正常なシステムでは無効なメッセージ本文は想定されないため、しきい値を 0 に設定することをおすすめします。

期間: 60

アラームを発生させるデータポイント数: 5

評価期間数: 5

比較演算子: GREATER_THAN_THRESHOLD

NumberOfNotificationsRedrivenToDlq

ディメンション: TopicName

アラームの説明: このアラームは、デッドレターキューに移動されたメッセージの数をモニタリングするのに役立ちます。

目的: このアラームは、デッドレターキューに移動されたメッセージを検出するのに使用されます。SNS が SQS、Lambda、または Firehose と連携している場合には、このアラームを作成することをお勧めします。

統計: Sum

推奨しきい値: 0.0

しきい値の根拠: どのサブスクライバータイプであっても、正常なシステムでは、メッセージはデッドレターキューに移動されないはずです。根本原因を特定して対処し、データ損失を防ぐためにデッドレターキュー内のメッセージを再処理できるように、メッセージがキューに入った場合は通知を受けることをお勧めします。

期間: 60

アラームを発生させるデータポイント数: 5

評価期間数: 5

比較演算子: GREATER_THAN_THRESHOLD

NumberOfNotificationsFailedToRedriveToDlq

ディメンション: TopicName

アラームの説明: このアラームは、デッドレターキューに移動できなかったメッセージをモニタリングするのに役立ちます。デッドレターキューが存在するかどうか、また正しく設定されているかどうかを確認します。また、SNS にデッドレターキューへのアクセス許可があることも確認します。詳細については、「デッドレターキューのドキュメント」を参照してください。

目的: このアラームは、デッドレターキューに移動できなかったメッセージを検出するのに使用されます。

統計: Sum

推奨しきい値: 0.0

しきい値の根拠: メッセージをデッドレターキューに移動できないのはほとんどの場合ミスです。しきい値の推奨値は 0 です。つまり、キューが設定されていると、処理に失敗したすべてのメッセージがデッドレターキューに到達できる必要があります。

期間: 60

アラームを発生させるデータポイント数: 5

評価期間数: 5

比較演算子: GREATER_THAN_THRESHOLD

SMSMonthToDateSpentUSD

ディメンション: TopicName

アラームの説明: このアラームは、SNS がメッセージを配信するのに十分なクォータがアカウントにあるかどうかをモニタリングするのに役立ちます。クォータに達すると、SNS は SMS メッセージを配信できなくなります。SMS の毎月の使用量クォータの設定、または AWS に対して使

用量クォータの引き上げをリクエストする方法については、「SMS メッセージプリファレンスを設定する」を参照してください。

目的: このアラームは、SMS メッセージを正常に配信するのに十分なクォータがアカウントにあるかどうかを検出するために使用されます。

統計: Maximum

推奨しきい値: 状況によって異なります。

しきい値の根拠: アカウントのクォータ (アカウントの使用量上限) に従ってしきい値を設定します。クォータ制限に達しようとしていることを早期に通知するしきい値を設定して、引き上げをリクエストする時間を確保してください。

期間: 60

アラームを発生させるデータポイント数: 5

評価期間数: 5

比較演算子: GREATER_THAN_THRESHOLD

SMSSuccessRate

ディメンション: TopicName

アラームの説明: このアラームは、SMS メッセージ配信の失敗率をモニタリングするのに役立ちます。CloudWatch Logs を設定して失敗の性質を把握し、それに基づいてアクションを実行できます。

目的: このアラームは、SMS メッセージ配信の失敗を検出するために使用されます。

統計: Average

推奨しきい値: 状況によって異なります。

しきい値の根拠: SMS メッセージ配信に失敗した場合の許容度に合わせて、アラームのしきい値を設定します。

期間: 60

アラームを発生させるデータポイント数: 5

評価期間数: 5

比較演算子: GREATER_THAN_THRESHOLD

Amazon SQS

ApproximateAgeOfOldestMessage

ディメンション: QueueName

アラームの説明: このアラームは、キュー内の最も古いメッセージの存続期間をモニタリングします。このアラームを使用して、コンシューマーが望ましい速度で SQS メッセージを処理しているかどうかをモニタリングできます。メッセージの存続期間を短縮するために、コンシューマー数またはコンシューマースループットを増やすことを検討してください。このメトリクスを `ApproximateNumberOfMessagesVisible` と組み合わせて使用すると、キューのバックログの大きさやメッセージの処理速度を判断できます。メッセージが処理前に削除されないようにするには、ポイズンピルの可能性があるメッセージを除外するようにデッドレターキューを設定することを検討してください。

目的: このアラームは、QueueName キュー内の最も古いメッセージの存続期間が長すぎるかどうかを検出するために使用されます。存続期間が長いということは、メッセージが十分に速く処理されていないか、またはキューに残っていて処理できないポイズンピルメッセージがあることを示している可能性があります。

統計: Maximum

推奨しきい値: 状況によって異なります。

しきい値の根拠: このアラームの推奨しきい値は、想定されるメッセージ処理時間によって大きく異なります。履歴データを使用して平均メッセージ処理時間を計算し、キューコンシューマーの SQS メッセージの最大予想処理時間よりも 50% 高く設定します。

期間: 60

アラームを発生させるデータポイント数: 15

評価期間数: 15

比較演算子: GREATER_THAN_OR_EQUAL_TO_THRESHOLD

ApproximateNumberOfMessagesNotVisible

ディメンション: QueueName

アラームの説明: このアラームは、QueueName に関する送信中メッセージが多いことを検出するのに役立ちます。トラブルシューティングのため、メッセージバックログの減少を確認してください。

目的: このアラームは、キューにある処理中のメッセージが多いことを検出するために使用されます。表示可能性がタイムアウトになるまでにコンシューマーがメッセージを削除しない場合、キューがポーリングされると、メッセージはキューに再び表示されます。FIFO キューでは、処理中のメッセージは最大 20,000 存在することが可能です。このクォータに達した場合、SQS はエラーメッセージを返しません。FIFOキューは、最初の 20k メッセージを検索して、使用可能なメッセージグループを判別します。つまり、1 つのメッセージグループにメッセージのバックログがある場合、バックログからそれらのメッセージを正常に使用するまで、後からキューに送信された他のメッセージグループからのメッセージを使用することはできません。

統計: Average

推奨しきい値: 状況によって異なります。

しきい値の根拠: このアラームの推奨しきい値は、想定される送信中メッセージ数によって大きく異なります。履歴データを使用して、想定される最大送信中メッセージ数を計算し、そのしきい値をこの値よりも 50% 高く設定します。キューのコンシューマーがメッセージを処理していても、キューからメッセージを削除していない場合、この数は突然増加します。

期間: 60

アラームを発生させるデータポイント数: 15

評価期間数: 15

比較演算子: GREATER_THAN_OR_EQUAL_TO_THRESHOLD

ApproximateNumberOfMessagesVisible

ディメンション: QueueName

アラームの説明: このアラームは、メッセージキューのバックログが予想よりも大きくなるのをモニタリングします。これは、コンシューマーが遅すぎるか、コンシューマーの数が足りないことを示しています。このアラームが ALARM 状態になった場合は、コンシューマー数を増やすか、コンシューマーの速度を上げることを検討してください。

目的: このアラームは、アクティブキューのメッセージ数が多すぎて、コンシューマーがメッセージを処理するのが遅いのか、またはメッセージを処理するのに十分なコンシューマーがいないのかを検出するために使用されます。

統計: Average

推奨しきい値: 状況によって異なります。

しきい値の根拠: 表示されるメッセージの数が予想外に多い場合は、コンシューマーがメッセージを想定した速度で処理していないことを示しています。このしきい値を設定するときは、履歴データを考慮する必要があります。

期間: 60

アラームを発生させるデータポイント数: 15

評価期間数: 15

比較演算子: GREATER_THAN_OR_EQUAL_TO_THRESHOLD

NumberOfMessagesSent

ディメンション: QueueName

アラームの説明: このアラームは、QueueName に対してプロデューサーからメッセージが送信されていないかどうかを検出するのに役立ちます。トラブルシューティングのため、プロデューサーがメッセージを送信していない理由を確認してください。

目的: このアラームは、プロデューサーがメッセージの送信を停止したことを検出するために使用されます。

統計: Sum

推奨しきい値: 0.0

しきい値の根拠: 送信されるメッセージの数が 0 の場合、プロデューサーはメッセージを一切送信していないことになります。このキューの TPS が低い場合は、それに応じて EvaluationPeriods の数値を増やしてください。

期間: 60

アラームを発生させるデータポイント数: 15

評価期間数: 15

比較演算子: LESS_THAN_OR_EQUAL_TO_THRESHOLD

AWS VPN

TunnelState

ディメンション: VpnId

アラームの説明: このアラームは、1 つまたは複数のトンネルの状態が DOWN になっているかどうかを把握するのに役立ちます。トラブルシューティングについては、「VPN トンネルのトラブルシューティング」を参照してください。

目的: このアラームは、この VPN のトンネルが 1 つでも DOWN 状態にあるかどうかを検出し、影響を受ける VPN をトラブルシューティングできるようにするために使用されます。このアラームは、トンネルが 1 つしか設定されていないネットワークでは常に ALARM 状態になります。

統計: Minimum

推奨しきい値: 1.0

しきい値の根拠: 1 未満の値は、少なくとも 1 つのトンネルが DOWN 状態であることを示します。

期間: 300

アラームを発生させるデータポイント数: 3

評価期間数: 3

比較演算子: LESS_THAN_THRESHOLD

TunnelState

ディメンション: TunnelIpAddress

アラームの説明: このアラームは、このトンネルの状態が DOWN になっているかどうかを把握するのに役立ちます。トラブルシューティングについては、「VPN トンネルのトラブルシューティング」を参照してください。

目的: このアラームは、トンネルが DOWN 状態かどうかを検出し、影響を受ける VPN をトラブルシューティングできるようにするために使用されます。このアラームは、トンネルが 1 つしか設定されていないネットワークでは常に ALARM 状態になります。

統計: Minimum

推奨しきい値: 1.0

しきい値の根拠: 1 未満の値は、トンネルが DOWN 状態であることを示します。

期間: 300

アラームを発生させるデータポイント数: 3

評価期間数: 3

比較演算子: LESS_THAN_THRESHOLD

メトリクスに関する警告

次のセクション内の手順では、メトリクスに CloudWatch アラームを作成する方法について説明します。

静的しきい値に基づいて CloudWatch アラームを作成する

アラームで監視する CloudWatch メトリクスと、このメトリクスのしきい値を選択します。指定した評価期間数にわたってメトリクスがしきい値を超えると、アラームが ALARM 状態に移行します。

CloudWatch のクロスアカウントオブザーバビリティでモニターリングアカウントとして設定されたアカウントでアラームを作成する場合、このモニターリングアカウントにリンクされたソースアカウントのメトリクスを監視するようにアラームを設定できます。詳細については、「CloudWatch のクロスアカウントオブザーバビリティ」を参照してください。

単一のメトリクスに基づいてアラームを作成するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで、[アラーム]、[すべてのアラーム] の順に選択します。
3. [アラームの作成] を選択します。
4. [メトリクスの選択] を選択します。
5. 次のいずれかを行います。
   - 必要なメトリクスが含まれているサービス名前空間を選択します。表示されるオプションを選択し続けて選択肢を絞り込みます。メトリクスのリストが表示されたら、必要なメトリクスの横にあるチェックボックスをオンにします。

- 検索ボックスに、メトリクス、アカウント ID、アカウントレベル、ディメンション、またはリソース ID の名前を入力します。次に、結果のいずれかを選択する操作を続けて、メトリクスを一覧表示します。必要なメトリクスの横にあるチェックボックスをオンにします。

6. [グラフ化したメトリクス] タブを選択します。

   a. [統計] で、統計や事前定義済みパーセンタイルのいずれかを選択するか、カスタムパーセンタイル (**p95.45** など) を指定します。

   b. [期間] で、アラームの評価期間を選択します。アラームを評価する場合、各期間は 1 つのデータポイントに集約されます。

      アラームの作成時に Y 軸の凡例を左側または右側に表示するかを選択することもできます。この設定が使用されるのは、アラームの作成時のみです。

   c. [メトリクスの選択] を選択します。

      [Specify metric and conditions (メトリクスと条件の指定)] ページに、選択したメトリクスと統計のグラフや他の情報が表示されます。

7. [Conditions (条件)] で、次のように指定します。

   a. [**#####**が次の時] で、メトリクスがしきい値より大きい、より小さい、またはしきい値と等しい必要があるかどうかを指定します。[than... (以下の値)] で、しきい値を指定します。

   b. [Additional configuration (追加設定)] を選択します。[Datapoints to alarm (アラームを発生させるデータポイント数)] で、アラームをトリガーするために ALARM 状態を維持する必要がある評価期間 (データポイント) の数を指定します。2 つの値が一致する場合は、該当する数の連続した期間でしきい値を超過したときに ALARM 状態に移行するアラームを作成します。

      N 個中 M 個のアラームを作成するには、2 番目の値よりも小さい数字を最初の値に指定します。詳細については、「アラームの評価」を参照してください。

   c. [Missing data treatment (欠落データの処理)] 、一部のデータポイントが欠落しているときのアラームによる対処方法を選択します。詳細については、「CloudWatch アラームの欠落データの処理の設定」を参照してください。

   d. モニターリングする統計としてパーセンタイルをアラームで使用している場合は、[サンプル数が少ないパーセンタイル] ボックスが表示されます。これを使用して、サンプル数が少ないケースを評価するか無視するかを選択します。[無視 (アラーム状態を維持する)] を選択すると、サンプル数が少なすぎる場合でも現在のアラーム状態が常に維持されます。詳細については、「パーセンタイルベースの CloudWatch アラームおよび少数のデータサンプル」を参照してください。

8. [Next (次へ)] を選択します。
9. [通知] で、アラームが ALARM 状態、OK 状態、または INSUFFICIENT_DATA 状態のときに通知するための SNS トピックを選択します。

   同じアラーム状態または複数の異なるアラーム状態について複数の通知を送信するには、[Add notification (通知の追加)] を選択します。

   CloudWatch のクロスアカウントオブザーバビリティで、通知を複数の AWS アカウントに送信するように選択できます。例えば、モニターリングアカウントとソースアカウント両方への送信です。

   アラームの通知を送信しない場合は、 [削除] を選択します。
10. アラームに伴って Auto Scaling、EC2、Lambda、または Systems Manager アクションを実行するには、該当するボタンを選択し、アラーム状態と実行するアクションを選択します。アラームは、ALARM 状態になったときにのみ、Systems Manager のアクションを実行できます。Systems Manager のアクションの詳細については、「アラームから OpsItems を作成するように CloudWatch を設定する」および「Incident creation」を参照してください。

    > **Note**
    >
    > SSM Incident Manager アクションを実行するアラームを作成するには、特定のアクセス許可が必要です。詳細については、AWS Systems Manager Incident Manager のアイデンティティベースのポリシーの例を参照してください。

11. 完了したら、[次へ] を選択します。
12. アラームの名前と説明を入力します。アラーム名には UTF-8 文字のみを使用する必要があり、ASCII 制御文字は使用できません。説明にはマークダウン形式を含めることができます。マークダウン形式は、CloudWatch コンソールのアラームの [詳細] タブにのみ表示されます。マークダウンは、ランブックや他の内部リソースへのリンクを追加するのに役立ちます。続いて、[次へ] を選択します。
13. [Preview and create (プレビューして作成)] で、情報と条件が正しいことを確認し、[アラームの作成] を選択します。

アラームはダッシュボードに追加することもできます。詳細については、「 CloudWatch ダッシュボードでアラームウィジェットを追加または削除する 」を参照してください。

メトリクス数式に基づく CloudWatch アラームの作成

メトリクスの数式に基づくアラームを作成するには、式で使用する 1 つ以上の CloudWatch メトリクスを選択します。次に、式、しきい値、および評価期間を指定します。

検索式に基づくアラームを作成することはできません。これは、検索式が複数の時系列を返し、数式に基づくアラームは 1 つの時系列しか監視できないためです。

メトリクス数式に基づくアラームを作成するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで、[Alarms] (アラーム) を選択し、[All alarms] (アラームの作成) を選択します。
3. [アラームの作成] を選択します。
4. [Select Metric] (メトリクスを選択) を選択してから、次のいずれかのアクションを実行します。
   - [AWS 名前空間] ドロップダウンまたは [カスタム名前空間] ドロップダウンから、名前空間を選択します。名前空間を選択した後、メトリクスのリストが表示されるまでオプションの選択を続けます。ここで、適切なメトリクスの横にあるチェックボックスを選択します。
   - 検索ボックスを使用して、メトリクス、アカウント ID、ディメンション、またはリソース ID を検索します。メトリクス、ディメンション、またはリソース ID を入力した後、メトリクスのリストが表示されるまでオプションを選択し続け、適切なメトリクスの横にあるチェックボックスをオンにします。
5. (オプション) メトリクス数式に別のメトリクスを追加する場合は、検索ボックスを使用して特定のメトリクスを検索できます。メトリクス数式には最大 10 個のメトリクスを追加できます。
6. [Graphed metrics] (グラフ化されたメトリクス) タブを選択します。以前に追加したメトリクスごとに、次のアクションを実行します。

   a. [Statistic] (統計) 列で、ドロップダウンメニューを選択します。ドロップダウンメニューで、定義済みのいずれかの統計値またはパーセンタイルを選択します。ドロップダウンメニューの検索ボックスを使用して、カスタムパーセンタイルを指定します。

   b. [Period] (期間) 列で、ドロップダウンメニューを選択します。ドロップダウンメニューで、定義済みのいずれかの評価期間を選択します。

      アラームの作成中に、Y 軸の凡例をグラフの左側と右側のいずれに表示するかを指定できます。

> **Note**
>
> CloudWatch がアラームを評価すると、期間は単一のデータポイントに集約されます。

7. [Add math] (数式を追加) ドロップダウンを選択し、事前定義されたメトリクス数式のリストから [Start with an empty expression] (空の式で開始) を選択します。

   [Start with an empty expression] (空の式で開始) を選択すると、数式を適用または編集する数式ボックスが表示されます。

8. 数式ボックスに数式を入力し、[Apply] (適用) を選択します。

   [Apply] (適用) を選択すると、[Label] (ラベル) 列の横に [ID] 列が表示されます。

   現在の数式の一部としてメトリクスまたは別のメトリクス数式の結果を使用するには、[ID] 列の下に表示されている値を使用します。[ID] の値を変更するには、現在の値の横にあるペンと紙のアイコンを選択します。新しい値の先頭は小文字にする必要があります。数字、文字、アンダースコア記号を使用できます。[ID] の値を重要な名前に変更すると、アラームのグラフがわかりやすくなります。

   Metric Math で使用できる関数の詳細については、「Metric Math 構文と関数」を参照してください。

9. (オプション) さらに数式を追加します。メトリクスおよび他の数式の結果の両方を新しい数式で使用できます。

10. アラームで使用する式を決めたら、ページの他の式やメトリクスごとに左のチェックボックスをオフにします。アラームで使用する式の横にあるチェックボックスのみオンにします。アラーム用に選択した式は、単一の時系列を生成し、グラフに 1 行のみを表示する必要があります。次に [メトリクスの選択] を選択します。

    [Specify metric and conditions (メトリクスと条件の指定)] ページに、選択した数式に関するグラフや他の情報が表示されます。

11. [**#**が次の時] で、式がしきい値より大きい、より小さい、またはしきい値と等しい必要があるかどうかを指定します。[than... (以下の値)] で、しきい値を指定します。

12. [Additional configuration (追加設定)] を選択します。[Datapoints to alarm (アラームを発生させるデータポイント数)] で、アラームをトリガーするために ALARM 状態を維持する必要がある評価期間 (データポイント) の数を指定します。2 つの値が一致する場合は、該当する数の連続した期間でしきい値を超過したときに ALARM 状態に移行するアラームを作成します。

N 個中 M 個のアラームを作成するには、2 番目の値よりも小さい数字を最初の値に指定します。詳細については、「アラームの評価」を参照してください。

13. [Missing data treatment (欠落データの処理)] 、一部のデータポイントが欠落しているときのアラームによる対処方法を選択します。詳細については、「CloudWatch アラームの欠落データの処理の設定」を参照してください。
14. [Next (次へ)] を選択します。
15. [通知] で、アラームが ALARM 状態、OK 状態、または INSUFFICIENT_DATA 状態のときに通知するための SNS トピックを選択します。

    同じアラーム状態または複数の異なるアラーム状態について複数の通知を送信するには、[Add notification (通知の追加)] を選択します。

    アラームの通知を送信しない場合は、 [削除] を選択します。
16. アラームに伴って Auto Scaling、EC2、Lambda、または Systems Manager アクションを実行するには、該当するボタンを選択し、アラーム状態と実行するアクションを選択します。Lambda 関数をアラームアクションとして選択する場合、関数名または ARN を指定し、オプションで関数の特定のバージョンを選択できます。

    アラームは、ALARM 状態になったときにのみ、Systems Manager のアクションを実行できます。Systems Manager のアクションの詳細については、「アラームから OpsItems を作成するように CloudWatch を設定する」および「Incident creation」を参照してください。

    > **Note**
    >
    > SSM Incident Manager アクションを実行するアラームを作成するには、特定のアクセス許可が必要です。詳細については、AWS Systems Manager Incident Manager のアイデンティティベースのポリシーの例を参照してください。
17. 完了したら、[次へ] を選択します。
18. アラームの名前と説明を入力します。次いで、[次へ] を選択します。

    アラーム名には UTF-8 文字のみを使用する必要があり、ASCII 制御文字は使用できません。説明にはマークダウン形式を含めることができます。マークダウン形式は、CloudWatch コンソールのアラームの [詳細] タブにのみ表示されます。マークダウンは、ランブックや他の内部リソースへのリンクを追加するのに役立ちます。

19. [Preview and create (プレビューして作成)] で、情報と条件が正しいことを確認し、[アラームの作成] を選択します。

アラームはダッシュボードに追加することもできます。詳細については、「CloudWatch ダッシュボードでアラームウィジェットを追加または削除する」を参照してください。

Metrics Insights クエリに基づく CloudWatch アラームの作成

単一の時系列を返す任意の Metrics Insights クエリに基づいてアラームを作成できます。この機能は、インフラストラクチャまたはアプリケーションのフリート全体で集計されたメトリクスを監視する動的アラームを作成する場合に特に役立ちます。一度作成されたアラームは、リソースがフリートに追加されたり、フリートから削除されたりしたときに調整されます。例えば、すべてのインスタンスの CPU 使用率を監視するアラームを作成すると、そのアラームはインスタンスの追加または削除に応じて動的に調整されます。

詳細な手順については、「Metrics Insights クエリでアラームを作成する」を参照してください。

接続されたデータソースに基づいてアラームを作成する

CloudWatch にないデータソースのメトリクスを監視するアラームを作成できます。これら以外のデータソースへの接続を作成する方法の詳細については、「他のデータソースにあるメトリクスへのクエリ」を参照してください。

接続しているデータソースのメトリクスに関するアラームを作成するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで、[Metrics]、[All metrics] を選択します。
3. [マルチソースクエリ] タブを選択します。
4. [データソース] で、使用するデータソースを選択します。
5. クエリビルダーから、アラームに使用するメトリクスを取得するためのクエリに必要な情報の入力を求められます。ワークフローはデータソースごとに異なり、データソースに合わせて調整されます。例えば、Amazon Managed Service for Prometheus と Prometheus データソースの場合、PromQL クエリエディタボックスがクエリヘルパーと共に表示されます。
6. クエリの作成が完了したら、[グラフクエリ] を選択します。
7. サンプルグラフが期待どおりになったら、[アラームの作成] を選択します。

8. [メトリクスと条件の指定] ページが表示されます。使用しているクエリが複数の時系列を生成する場合は、ページの上部に警告バナーが表示されます。その場合は、[集計関数] で時系列の集計に使用する関数を選択します。
9. (オプション) アラームの [ラベル] を追加します。
10. [***your-metric-name*** が次の時...] で、[より大きい]、[以上]、[以下]、または [より低い] から選択します。次に、[... よりも] で、しきい値の数値を指定します。
11. [Additional configuration (追加設定)] を選択します。[Datapoints to alarm (アラームを発生させるデータポイント数)] で、アラームをトリガーするために ALARM 状態を維持する必要がある評価期間 (データポイント) の数を指定します。2 つの値が一致する場合は、該当する数の連続した期間でしきい値を超過したときに ALARM 状態に移行するアラームを作成します。

    N 個中 M 個のアラームを作成するには、2 番目の値よりも小さい数字を最初の値に指定します。(詳細については、アラームの評価 を参照してください)。
12. [Missing data treatment] (欠落データの処理) で、一部のデータポイントが欠落している際のアラームによる対処方法を選択します。詳細については、「CloudWatch アラームの欠落データの処理の設定」を参照してください。
13. [Next] を選択します。
14. [通知] で、アラームが ALARM、OK、または INSUFFICIENT_DATA 状態に遷移するときに通知するための Amazon SNS トピックを指定します。

    a. (オプション) 同じアラーム状態または異なるアラーム状態について複数の通知を送信するには、[Add notification] (通知の追加)を選択します。

    > **Note**
    >
    > [アラーム] 状態になったときだけでなく、[データ不足] 状態になったときにもアクションを実行するようにアラームを設定することをお勧めします。これは、データソースに接続する Lambda 関数に関する多くの問題により、アラームが [データ不足] に遷移する可能性があるためです。

    b. (オプション) Amazon SNS の通知を送信しない場合は、[削除] を選択します。
15. アラームに伴って Auto Scaling、EC2、Lambda、または Systems Manager アクションを実行するには、該当するボタンを選択し、アラーム状態と実行するアクションを選択します。Lambda 関数をアラームアクションとして選択する場合、関数名または ARN を指定し、オプションで関数の特定のバージョンを選択できます。

アラームは、ALARM 状態になったときにのみ、Systems Manager のアクションを実行できます。Systems Manager のアクションの詳細については、「アラームから OpsItems を作成するように CloudWatch を設定する」および「Incident creation」を参照してください。

> **Note**
>
> SSM Incident Manager アクションを実行するアラームを作成するには、特定のアクセス許可が必要です。詳細については、AWS Systems Manager Incident Manager のアイデンティティベースのポリシーの例を参照してください。

16. [Next] を選択します。
17. [Name and description] (名前と説明) にアラームの名前と説明を入力し、[Next] (次へ) をクリックします。アラーム名には UTF-8 文字のみを使用する必要があり、ASCII 制御文字は使用できません。説明にはマークダウン形式を含めることができます。マークダウン形式は、CloudWatch コンソールのアラームの [詳細] タブにのみ表示されます。マークダウンは、ランブックや他の内部リソースへのリンクを追加するのに役立ちます。

> **Tip**
>
> アラーム名には UTF-8 文字のみを使用する必要があります。ASCII 制御文字を含めることはできません。

18. [Preview and create] (プレビューと作成) で、アラームの情報と条件が正しいことを確認し、[Create alarm] (アラームの作成) をクリックします。

接続されているデータソースのアラームに関する詳細

- CloudWatch はアラームを評価するとき、アラームの期間が 1 分を超えていても、1 分ごとにアラームを評価します。アラームが機能するためには、Lambda 関数が期間の長さの倍数だけでなく、任意の分から始まるタイムスタンプのリストを返すことができなければなりません。これらのタイムスタンプは、1 つの期間の長さの間隔を空ける必要があります。

  したがって、Lambda によってクエリされたデータソースが期間の長さの倍数のタイムスタンプしか返せない場合、関数は取得したデータを「再サンプリング」して、`GetMetricData` リクエストで期待されるタイムスタンプと一致させる必要があります。

例えば、5 分間隔のアラームは、毎回 1 分ずつシフトする 5 分のウィンドウを使用して 1 分ごとに評価されます。この場合は以下のようになります。

- 12:15:00 のアラーム評価では、CloudWatch は 12:00:00、12:05:00、12:10:00 のタイムスタンプを持つデータポイントを予想します。
- 次に、12:16:00 のアラーム評価では、CloudWatch は 12:01:00、12:06:00、12:11:00 のタイムスタンプを持つデータポイントを予想します。

- CloudWatch がアラームを評価すると、Lambda 関数によって返されるデータポイントのうち、予想されるタイムスタンプと一致しないものはすべて削除され、残りの予想されるデータポイントを使用してアラームが評価されます。例えば、12:15:00 でアラームが評価されると、12:00:00、12:05:00、12:10:00 のタイムスタンプを持つデータを予想します。12:00:00、12:05:00、12:06:00、12:10:00 のタイムスタンプを持つデータを受信すると、12:06:00 からのデータは削除され、CloudWatch は他のタイムスタンプを使用してアラームを評価します。

  次の 12:16:00 での評価では、12:01:00、12:06:00、12:11:00 のタイムスタンプを持つデータを予想します。タイムスタンプが 12:00:00、12:05:00、12:10:00 のデータしかない場合、これらのデータポイントはすべて 12:16:00 で無視され、アラームは、指定されている欠落データの処理方法に従って状態に遷移します。詳細については、「アラームの評価」を参照してください。
- いくつかの Lambda 関数の失敗ユースケースでは、設定されているアラームによる欠落データの処理方法に関係なく、アラームが INSUFFICIENT_DATA に遷移するため、これらのアラームを作成して、INSUFFICIENT_DATA 状態に遷移したときにアクションを実行することをお勧めします。
- Lambda 関数が エラーを返すか、部分的なデータを返す場合:
  - Lambda 関数の呼び出しでアクセス権限の問題が発生した場合、アラームの作成時に欠落データを処理するように指定した方法に従って、アラームは欠落データの遷移を開始します。
  - Lambda 関数が 'StatusCode' = 'PartialData' を返した場合、アラームの評価は失敗し、3 回試行した後、アラームは INSUFFICIENT_DATA に遷移します。これには約 3 分かかります。
  - Lambda 関数から発生するその他の エラーにより、アラームは INSUFFICIENT_DATA に遷移します。
- Lambda 関数によってリクエストされたメトリクスに遅延があり、最後のデータポイントが常に欠落している場合は、回避策を使用する必要があります。N 個中 M 個のアラームを作成するか、ア

ラームの評価期間を長くすることができます。N 個中 M 個のアラームについては、「アラームの評価」を参照してください。

異常検出に基づいて CloudWatch アラームを作成する

CloudWatch の異常検出に基づいてアラームを作成できます。これにより、過去のメトリクスデータが分析され、予想される値のモデルが作成されます。予想される値では、メトリクスの一般的な時間単位、日単位、週単位のパターンを考慮します。

異常検出のしきい値を設定すると、CloudWatch は、このしきい値をモデルで使用して、メトリクスの「正常」な値の範囲を決定します。しきい値を高くするほど、「正常」な値の範囲が広がります。

アラームがトリガーされるのが、メトリック値が想定値の範囲を上回る場合、下回る場合、または上回るか下回った場合のいずれかを選択できます。

また、単一のメトリクスとメトリクスの数式の出力で異常検出アラームを作成することもできます。これらの式を使用して、異常検出バンドを可視化するグラフを作成できます。

CloudWatch クロスアカウントオブザーバビリティのモニタリングアカウントとして設定されたアカウントでは、モニタリングアカウントのメトリクスに加え、ソースアカウントのメトリクスに対しても異常ディテクタを作成できます。

詳細については、「CloudWatch 異常検出の使用」を参照してください。

> **Note**
>
> メトリクスコンソールで、既に可視化目的で異常検出を使用しているメトリクスに異常検出アラームを作成する場合、アラームに設定したしきい値により、既に可視化に使用されているしきい値が変更されることはありません。詳細については、「グラフの作成」を参照してください。

異常検出に基づくアラームを作成するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで、[アラーム]、[すべてのアラーム] の順に選択します。
3. [アラームの作成] を選択します。
4. [メトリクスの選択] を選択します。

5. 次のいずれかを行います。
   - メトリクスを含むサービス名前空間を選択し、表示されるオプションを順次選択して、オプションを絞り込みます。メトリクスのリストが表示されたら、メトリクスの横にあるチェックボックスをオンにします。
   - 検索ボックスに、メトリクス、ディメンション、またはリソース ID の名前を入力します。結果のいずれかを選択してから、メトリクスのリストが表示されるまで、表示されるオプションを選択する操作を続けます。メトリクスの横にあるチェックボックスをオンにします。
6. [グラフ化したメトリクス] を選択します。
   a. (オプション) [Statistic] (統計) 列でドロップダウンを選択してから、事前定義済みの統計またはパーセンタイルのいずれかを選択します。ドロップダウンの検索ボックスを使用して、「**p95.45**」などのカスタムパーセンタイルを指定できます。
   b. (オプション) [Period] (期間) 列でドロップダウンを選択してから、事前定義済みの評価期間のいずれかを選択します。

   > **Note**
   > CloudWatch がアラームを評価すると、その期間が単一のデータポイントに集約されます。異常検出のアラームの場合、評価期間は 1 分以上である必要があります。

7. [Next] を選択します。
8. [Conditions (条件)] で、次のように指定します。
   a. [異常検出] を選択します。

   このメトリクスと統計のモデルが既に存在する場合、CloudWatch では、画面上部のサンプルグラフに異常検出バンドのプレビューが表示されます。アラームを作成した後、グラフに実際の異常検出バンドが表示されるまでに最大 15 分かかる場合があります。その前に、表示されるバンドは異常検出バンドの近似値です。

   > **Tip**
   > グラフを表示する時間枠を長くするには、ページの右上にある [Edit] (編集) を選択します。

このメトリクスと統計のモデルがまだ存在しない場合、CloudWatch では、アラームの作成が完了した後に異常検出バンドが生成されます。新しいモデルの場合、実際の異常検出バンドがグラフに表示されるまでに最大 3 時間かかる場合があります。新しいモデルのトレーニングには最大 2 週間かかる場合があり、これにより異常検出バンドはより正確な予測値を示すようになります。

b. [Whenever ***metric*** is] (メトリクスが次の時) で、アラームをトリガーするタイミングを指定します。例えば、メトリクスがバンドを超えている、下回っている、または (いずれかの方向に) 外れている場合です。

c. [Anomaly detection threshold (異常検出のしきい値)] で、異常検出のしきい値に使用する数値を選択します。数値を大きくすると、メトリクスの変化に対してより高い許容度をもつ、「通常の」値の広いバンドが作成されます。数値を小さくすると、メトリクスの偏差が小さい、ALARM 状態の細いバンドが作成されます。数値を整数にする必要はありません。

d. [Additional configuration (追加設定)] を選択します。[Datapoints to alarm (アラームを発生させるデータポイント数)] で、アラームをトリガーするために ALARM 状態を維持する必要がある評価期間 (データポイント) の数を指定します。2 つの値が一致する場合は、該当する数の連続した期間でしきい値を超過したときに ALARM 状態に移行するアラームを作成します。

N 個中 M 個のアラームを作成するには、2 番目の値よりも小さい数字を最初の値に指定します。(詳細については、アラームの評価 を参照してください)。

e. [Missing data treatment] (欠落データの処理) で、一部のデータポイントが欠落している際のアラームによる対処方法を選択します。(詳細については、CloudWatch アラームの欠落データの処理の設定 を参照してください)。

f. モニターリングする統計としてパーセンタイルをアラームで使用している場合は、[サンプル数が少ないパーセンタイル] ボックスが表示されます。これを使用して、サンプル数が少ないケースを評価するか無視するかを選択します。[Ignore (maintain alarm state)] (無視 (アラーム状態を維持する)) を選択すると、サンプル数が少なすぎる場合でも現在のアラーム状態が常に維持されます。詳細については、「パーセンタイルベースの CloudWatch アラームおよび少数のデータサンプル」を参照してください。

9. [Next (次へ)] を選択します。

10. [通知] で、アラームが ALARM 状態、OK 状態、または INSUFFICIENT_DATA 状態のときに通知するための SNS トピックを選択します。

同じアラーム状態または異なるアラーム状態について複数の通知を送信するには、[Add notification] (通知の追加)を選択します。

アラームの通知を送信しない場合は、[Remove] (削除) を選択します。

11. 状態が変更されたときに EC2 アクションを実行したり、Lambda 関数を呼び出したりするように、あるいは、ALARM 状態になったときに Systems Manager の OpsItem またはインシデントを作成するようにアラームを設定できます。これを行うには、該当するボタンをクリックし、アラーム状態および実行するアクションを選択します。

    Lambda 関数をアラームアクションとして選択する場合、関数名または ARN を指定し、オプションで関数の特定のバージョンを選択できます。

    Systems Manager のアクションの詳細については、「アラームから OpsItems を作成するように CloudWatch を設定する」および「Incident creation」を参照してください。

    > **Note**
    >
    > AWS Systems Manager Incident Manager アクションを実行するアラームを作成するには、特定のアクセス許可が必要です。詳細については、AWS Systems Manager Incident Manager のアイデンティティベースのポリシーの例を参照してください。

12. [Next] を選択します。
13. [Name and description] (名前と説明) にアラームの名前と説明を入力し、[Next] (次へ) をクリックします。アラーム名には UTF-8 文字のみを使用する必要があり、ASCII 制御文字は使用できません。説明にはマークダウン形式を含めることができます。マークダウン形式は、CloudWatch コンソールのアラームの [詳細] タブにのみ表示されます。マークダウンは、ランブックや他の内部リソースへのリンクを追加するのに役立ちます。

    > **Tip**
    >
    > アラーム名には UTF-8 文字のみを使用する必要があり、ASCII 制御文字は使用できません

14. [Preview and create] (プレビューと作成) で、アラームの情報と条件が正しいことを確認し、[Create alarm] (アラームの作成) をクリックします。

異常検出モデルの変更

アラームの作成後、異常検出モデルを調整できます。モデルの作成に使用しない特定の期間を除外できます。システムの停止、デプロイ、休日などの異常なイベントは、トレーニングデータから除外することが重要です。夏時間の変更に合わせてモデルを調整するかどうかも指定できます。

アラームの異常検出モデルを調整するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで、[アラーム]、[すべてのアラーム] の順に選択します。
3. アラーム名を指定します。必要に応じて検索ボックスを使用し、アラームを見つけます。
4. [分析]、[メトリクス内] を選択します。
5. [詳細] 列で、[ANOMALY_DETECTION_BAND]、[異常検出モデルを編集] を選択します。
6. モデルの生成に利用される期間からある期間を除外するには、[終了日] の横にあるカレンダーアイコンを選択します。次に、トレーニングから除外する日付と時刻を選択または入力して、[適用] をクリックします。
7. メトリクスが夏時間の変更に影響を受ける場合は、[Metric timezone] (メトリクスタイムゾーン) ボックスで、適切なタイムゾーンを選択します。
8. [Update] (更新) を選択します。

異常検出モデルの削除

アラームに異常検出を使用すると、 料金が発生します。ベストプラクティスとして、アラームで異常検出モデルが不要になった場合は、最初にアラームを削除し、2 番目にモデルを削除します。異常検出アラームが評価されると、欠落している異常ディテクタがユーザーに代わって作成されます。アラームを削除しないでモデルを削除すると、アラームによりモデルが自動的に再作成されます。

アラームを削除するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで、[Alarms] (アラーム)、[All Alarms] (すべてのアラーム) の順に選択します。
3. アラーム名を指定します。
4. [アクション]、[削除] の順に選択します。
5. 確認ボックスで [Delete] (削除) を選択します。

アラームに使用された異常検出モデルを削除するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで、[Metrics] (メトリクス)、[All metrics] (すべてのメトリクス) の順に選択します。
3. [Browse] (参照) を選択して、異常検出モデルを含むメトリクスを選択します。メトリクスは、検索ボックスで検索するか、オプションから選択することができます。
   - (オプション) 元のインターフェイスを使用している場合は、[All metrics] (すべてのメトリクス) を選択して、異常検出モデルを含むメトリクスを選択します。メトリクスは、検索ボックスでを検索するか、オプションから選択することができます。
4. [グラフ化したメトリクス] を選択します。
5. [Graphed metrics] (グラフ化したメトリクス) タブで、削除したい異常検知モデルの名前を選択し、[Delete anomaly detection model] (異常検出モデルを削除) を選択します。
   - (オプション) 元のインターフェイスを使用している場合は、[Edit model] (モデルの編集) を選択します。新しい画面が表示されます。新しい画面で、[Delete model] (モデルの削除) を選択して [Delete] (削除) をクリックします。

ログに対するアラーム

次のセクション内の手順では、ログに対する CloudWatch アラームを作成する方法について説明します。

ロググループのメトリクスフィルターに基づく CloudWatch アラームの作成

このセクションの手順では、ロググループのメトリクスフィルターに基づいてアラームを作成する方法について説明します。メトリクスフィルターを使用すると、データが CloudWatch に送信された時に、ログデータの用語やパターンを検索することができます。詳細については、「Amazon CloudWatch Logs ユーザーガイド」の「フィルターを使用したログイベントからのメトリクスの作成」を参照してください。ロググループのメトリクスフィルターに基づいてアラームを作成する前に、次のアクションを完了する必要があります。

- ロググループを作成します。詳細については、「Amazon CloudWatch Logs ユーザーガイド」の「ロググループとログストリームの操作」を参照してください。
- メトリクスのフィルターを作成します。詳細については、「Amazon CloudWatch Logs ユーザーガイド」の「ロググループのメトリクスフィルターの作成」を参照してください。

ロググループのメトリクスフィルターに基づいてアラームを作成するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで、[Logs] (ログ)、[Log groups] (ロググループ) の順に選択します。
3. メトリクスフィルターを含むロググループを選択します。
4. [Metric filters] (メトリクスフィルター) を選択します。
5. メトリクスフィルタータブで、アラームのベースにするメトリクスフィルターのボックスを選択します。
6. [アラームの作成] を選択します。
7. (オプション) [Metric] (メトリクス) で、[Metric name] (メトリクス名)、[Statistic] (統計)、および [Period] (期間) を編集します。
8. [Conditions (条件)] で、次のように指定します。
   a. [Threshold type] (しきい値の種類) で、[Static] (静的) または [Anomaly detection].(異常検出) を選択します。
   b. [Whenever ***your-metric-name*** is. . .] (your-metric-name が次の場合) で、[Greater] (より大きい)、[Greater/Equal] (以上)、[Lower/Equal] (以下) または [Lower] (より低い) から選択します。
   c. [than . . .] (次と比較) の場合は、しきい値の数値を指定します。
9. [Additional configuration (追加設定)] を選択します。
   a. [Data points to alarm] (アラームを発生させるデータポイント) で、ALARM 状態に移行するアラームをトリガーするデータポイントの数を指定します。一致する値を指定した場合、該当する連続した期間でしきい値を超過すると、アラームが ALARM 状態に移行します。N 個中 M 個のアラームを作成するには、2 番目の値よりも小さい数字を最初の値に指定します。詳細については、「Using Amazon CloudWatch alarms」(Amazon CloudWatch アラームを使用する) を参照してください。
   b. [Missing data treatment] (欠落データの処理) で、アラームが評価される際の欠落データの処理方法を指定するオプションを選択します。
10. [Next] を選択します。
11. [Notification] (通知) で、アラームが ALARM、OK、または INSUFFICIENT_DATA 状態の時に通知するための Amazon SNS トピックを指定します。
    a. (オプション) 同じアラーム状態または異なるアラーム状態について複数の通知を送信するには、[Add notification] (通知の追加)を選択します。

b. (オプション) アラームの通知を送信しない場合は、[削除] を選択します。

12. アラームに伴って Auto Scaling、EC2、Lambda、または Systems Manager アクションを実行するには、該当するボタンを選択し、アラーム状態と実行するアクションを選択します。Lambda 関数をアラームアクションとして選択する場合、関数名または ARN を指定し、オプションで関数の特定のバージョンを選択できます。

    アラームは、ALARM 状態になったときにのみ、Systems Manager のアクションを実行できます。Systems Manager のアクションの詳細については、「アラームから OpsItems を作成するように CloudWatch を設定する」および「Incident creation」を参照してください。

    > **Note**
    >
    > SSM Incident Manager アクションを実行するアラームを作成するには、特定のアクセス許可が必要です。詳細については、AWS Systems Manager Incident Manager のアイデンティティベースのポリシーの例を参照してください。

13. [Next] を選択します。
14. [名前と説明] にアラームの名前と説明を入力します。アラーム名には UTF-8 文字のみを使用する必要があり、ASCII 制御文字は使用できません。説明にはマークダウン形式を含めることができます。マークダウン形式は、CloudWatch コンソールのアラームの [詳細] タブにのみ表示されます。マークダウンは、ランブックや他の内部リソースへのリンクを追加するのに役立ちます。
15. [Preview and create] (プレビューと作成) で、設定が正しいことを確認し、[Create alarm] (アラームの作成) を選択します。

アラームの組み合わせ

CloudWatch では、複数のアラームを 1 つの複合アラームにまとめて、アプリケーション全体またはリソースグループ全体にわたってまとめて集計されたヘルスインジケーターを作成できます。複合アラームとは、他のアラームの状態をモニタリングして自身の状態を判定するアラームです。ブール論理を使用して、モニタリング対象アラームのステータスを組み合わせるルールを定義します。

複合アラームを使用すると、集約したレベルでのみアクションを実行すればよいので、アラームのノイズを低減できます。例えば、ウェブサーバーに関連するアラームがトリガーされた場合にウェブサーバーチームに通知を送信する複合アラームを作成できます。これらのアラームのいずれかが ALARM 状態になると、複合アラームは ALARM 状態になり、チームに通知を送信します。ウェブサーバーに関連する他のアラームが ALARM 状態になっても、複合アラームは現状を既に通知しているので、チームが新しい通知で過負荷になることはありません。

また、複合アラームを使用して複雑なアラーム条件を作成し、さまざまな条件が満たされた場合にのみアクションを実行することもできます。例えば、CPU アラームとメモリアラームを組み合わせた複合アラームを作成して、CPU アラームとメモリアラームの両方がトリガーされた場合にのみチームに通知することができます。

複合アラームを使用する

複合アラームを作成するには、次の 2 つの方法があります。

- 複合アラームレベルでのみ実行したいアクションを設定し、基礎となるモニタリング対象アラームはアクションなしで作成する
- 複合アラームレベルでは異なるアクションのセットを設定する。例えば、問題が広範囲に及ぶ場合、複合アラームアクションには別のチームを従事させることができます。

複合アラームでは次のアクションのみを実行できます。

- Amazon SNS トピックへの通知
- Lambda 関数を呼び出す
- Systems Manager Ops Center での OpsItems の作成
- Systems Manager Incident Manager でのインシデントの作成

> **Note**
>
> 複合アラームの基盤となるすべてのアラームは、複合アラームと同じアカウントおよびリージョンに存在する必要があります。ただし、CloudWatch のクロスアカウントオブザーバビリティモニターリングアカウントで複合アラームを設定した場合、基盤となるアラームが、異なるソースアカウントとモニターリングアカウント自体のメトリクスを監視できます。詳細については、「CloudWatch のクロスアカウントオブザーバビリティ」を参照してください。
>
> 1 つの複合アラームで 100 の基盤となるアラームを監視でき、1 つのアラームは 150 の複合アラームで監視できます。

ルール式

すべての複合アラームには、ルール式が含まれています。ルール式は、複合アラームに監視および状態を判別するアラームを指示するものです。ルール式では、メトリクスアラームと複合アラームを参

照できます。ルール式でアラームを参照するときは、アラームが次の 3 つのうちのどの状態になるかを決定する関数をアラームに指定します。

- アラーム

  指定されたアラームが ALARM 状態である場合、ALARM (「alarm-name または alarm-ARN」) は TRUE になります。
- OK

  指定されたアラームが OK 状態である場合、OK (「alarm-name または alarm-ARN」) は TRUE になります。
- INSUFFICIENT_DATA

  指定されたアラームが INSUFFICIENT_DATA 状態である場合、INSUFFICIENT_DATA (「alarm-name または alarm-ARN」) は TRUE になります。

> **Note**
>
> TRUE は常に TRUE と評価され、FALSE は常に FALSE と評価されます。

式の例

リクエストパラメータ `AlarmRule` は、論理演算子 `AND`、`OR`、および `NOT` の使用をサポートしています。これにより、複数の関数を 1 つの式に組み合わせることができます。次の式の例は、複合アラームで基盤となるアラームを設定する方法を示しています。

- `ALARM(CPUUtilizationTooHigh) AND ALARM(DiskReadOpsTooHigh)`

  この式は、`CPUUtilizationTooHigh` および `DiskReadOpsTooHigh` が `ALARM` 状態である場合のみ、複合アラームが `ALARM` 状態になることを指定しています。
- `ALARM(CPUUtilizationTooHigh) AND NOT ALARM(DeploymentInProgress)`

  この式は、`CPUUtilizationTooHigh` が `ALARM` 状態で、`DeploymentInProgress` が `ALARM` 状態ではない場合、複合アラームが `ALARM` 状態になることを指定しています。これは、デプロイ期間中にアラームのノイズを低減する複合アラームの例です。
- `(ALARM(CPUUtilizationTooHigh) OR ALARM(DiskReadOpsTooHigh)) AND OK(NetworkOutTooHigh)`

この式は、(ALARM(CPUUtilizationTooHigh) または (DiskReadOpsTooHigh) が ALARM 状態で、(NetworkOutTooHigh) が OK 状態である場合、複合アラームが ALARM 状態になることを指定しています。これは、ネットワークの問題が発生している間、基盤となるアラームのいずれかが ALARM 状態ではない場合に通知を送信しないことで、アラームのノイズを低減する複合アラームの例です。

トピック

複合アラームを作成する

このセクションのステップでは、CloudWatch コンソールを使用して複合アラームを作成する方法について説明します。API または AWS CLI を使用して複合アラームを作成することもできます。詳細については、PutCompositeAlarm または put-composite-alarm を参照してください。

複合アラームを作成するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで、[Alarms] (アラーム) を選択し、[All alarms] (アラームの作成) を選択します。
3. アラームのリストで、ルール式で参照する既存のアラームの横にあるチェックボックスをオンにしてから、[Create composite alarm] (複合アラームの作成) を選択します。
4. [Specify composite alarm conditions] (複合アラーム条件を指定) で、新しい複合アラームのルール式を指定します。

> **Note**
>
> [Conditions] (条件) ボックスに、アラームのリストから選択したアラームが自動的に表示されます。デフォルトでは、ALARM 関数が各アラームに指定されており、各アラームは論理演算子 OR によって結合されています。

次のサブステップに従って、ルール式を変更できます。

a. 各アラームで必要な状態は、ALARM から OK または INSUFFICENT_DATA に変更できます。

b. ルール式の論理演算子は、OR から AND または NOT に変更できます。また、関数をグループ化するための括弧を追加できます。

c. ルール式に他のアラームを含めることも、ルール式からアラームを削除することもできます。

例: 条件付きのルール式

```
(ALARM("CPUUtilizationTooHigh") OR
ALARM("DiskReadOpsTooHigh")) AND
OK("NetworkOutTooHigh")
```

このルール式の例では、OK("NetworkOutTooHigh") が OK であると同時に ALARM ("CPUUtilizationTooHigh" or ALARM("DiskReadOpsTooHigh") が ALARM 状態であるとき、複合アラームが ALARM 状態になります。

5. 完了したら、[次へ] を選択します。

6. [Configure actions] (アクションの設定) で、次を選択できます。

[Notification] (通知) から

- [Select an exisiting SNS topic] (既存の SNS トピックを選択)、[Create a new SNS topic] (新しい SNS トピックを作成)、または [Use a topic ARN] (トピック APN を使用) を選択すると、通知を受け取る SNS トピックを定義できます。
- [Add notification] (通知の追加) を選択すると、アラームで同じアラーム状態または異なるアラーム状態についての複数の通知を送信できます。
- [Remove] (削除) を選択すると、アラームによる通知の送信やアクションの実行を停止できます。

(オプション) 状態が変化したときにアラームが Lambda 関数を呼び出すようにするには、[Lambda アクションの追加] を選択します。次に、関数名または ARN を指定し、オプションで関数の特定のバージョンを選択します。

[Systems Manager action] (Systems Manager のアクション) から

- [Add Systems Manager action] (Systems Manager アクションを追加する) を選択すると、アラームが ALARM 状態になったときに SSM アクションを実行できます。

  Systems Manager のアクションの詳細については、「AWS Systems Manager ユーザーガイド」の「アラームから OpsItems を作成するように CloudWatch を設定する」、「Incident Manager ユーザーガイド」の「Incident creation」(インシデントの作成) を参照してください。SSM Incident Manager アクションを実行するアラームを作成するには、正しいアクセス許可が必要です。詳細については、「Incident Manager ユーザーガイド」の「AWS Systems Manager Incident Manager におけるアイデンティティベースのポリシーの例」を参照してください。

7. 完了したら、[次へ] を選択します。
8. [Add name and description] (名前と説明を追加) で、新しい複合アラームのアラーム名とオプションの説明を入力します。アラーム名には UTF-8 文字のみを使用する必要があり、ASCII 制御文字は使用できません。説明にはマークダウン形式を含めることができます。マークダウン形式は、CloudWatch コンソールのアラームの [詳細] タブにのみ表示されます。マークダウンは、ランブックや他の内部リソースへのリンクを追加するのに役立ちます。
9. 完了したら、[次へ] を選択します。
10. [Preview and create] (プレビューと作成) で情報を確認し、[Create composite alarm] (複合アラームの作成) を選択します。

> **Note**
>
> 1 つの複合アラームと別の複合アラームが互いに依存する複合アラームのサイクルを作成できます。このシナリオの場合、複合アラームは評価されなくなります。また、複合アラームは互いに依存しているため削除できません。複合アラーム間のサイクルの依存関係を断ち切る最も簡単な方法は、複合アラームのうちの 1 つで、関数 `AlarmRule` を `False` に変更することです。

複合アラームのアクションの抑制

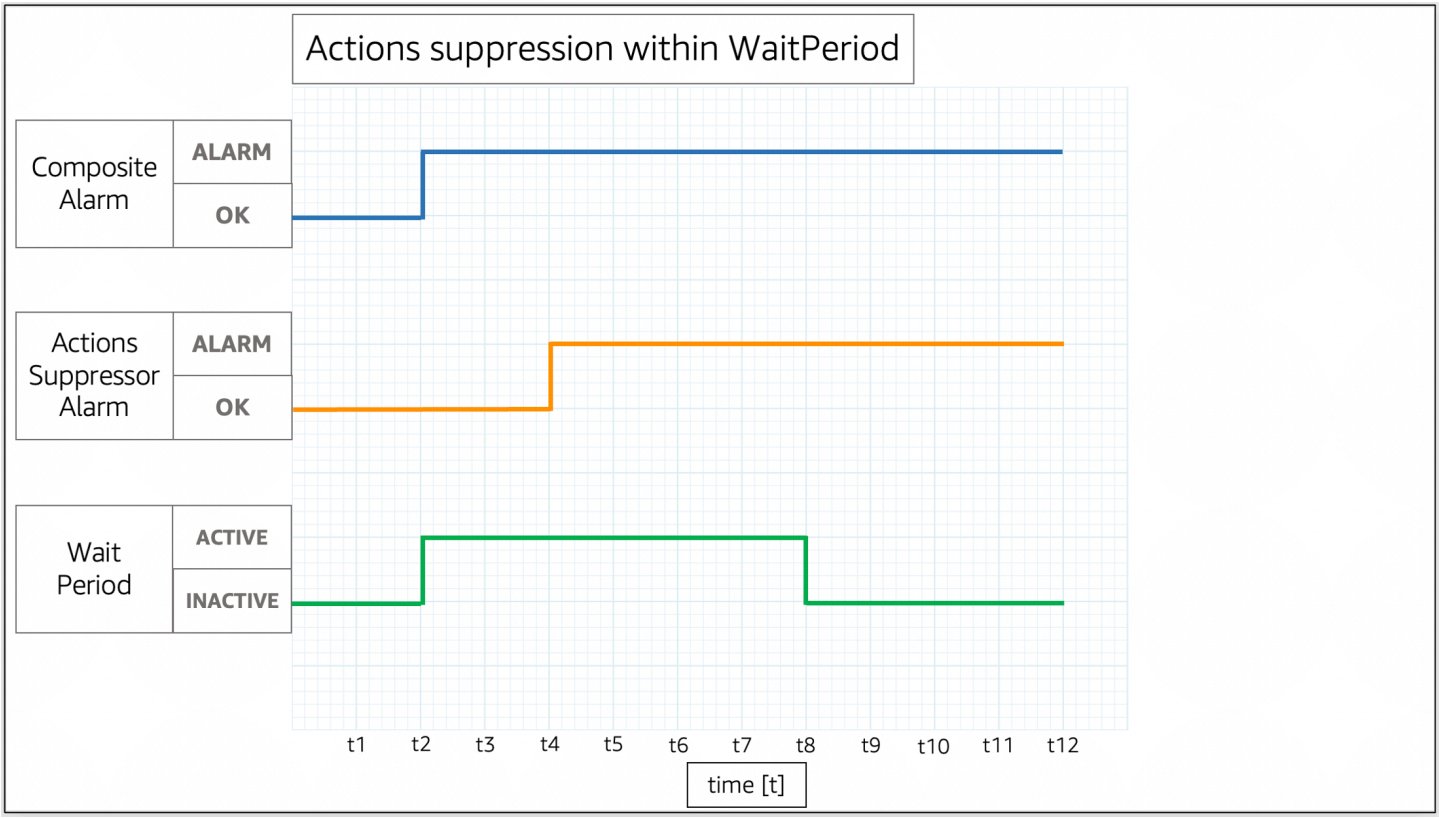
複合アラームでは複数のアラーム全体で健全性を集約して把握できるため、それらのアラームがトリガーされることが予想される状況はよくあります。例えば、アプリケーションのメンテナンス期間中や、進行中のインシデントを調査するときなどです。このような状況では、不要な通知や新しいインシデントチケットが生成されないように、複合アラームのアクションを抑制したい場合があります。

複合アラームのアクション抑制では、アラームをサプレッサーアラームとして定義します。サプレッサーアラームは、複合アラームでアクションが実行されるのを防ぐアラームです。例えば、サポートしているリソースのステータスを表すサプレッサーアラームを指定できます。サポートしているリソースがダウンしている場合、サプレッサーアラームは複合アラームによる通知の送信を妨げます。複合アラームのアクション抑制は、アラームのノイズを低減するのに役立ちます。そのため、アラームの管理に費やす時間を減らし、オペレーションに集中する時間を増やすことができます。

サプレッサーアラームは、複合アラームを設定する際に指定します。すべてのアラームは、サプレッサーアラームとして機能できます。サプレッサーアラームの状態が OK から ALARM に変わると、その複合アラームはアクションを停止します。サプレッサーアラームの状態が ALARM から OK に変わると、その複合アラームはアクションを再開します。

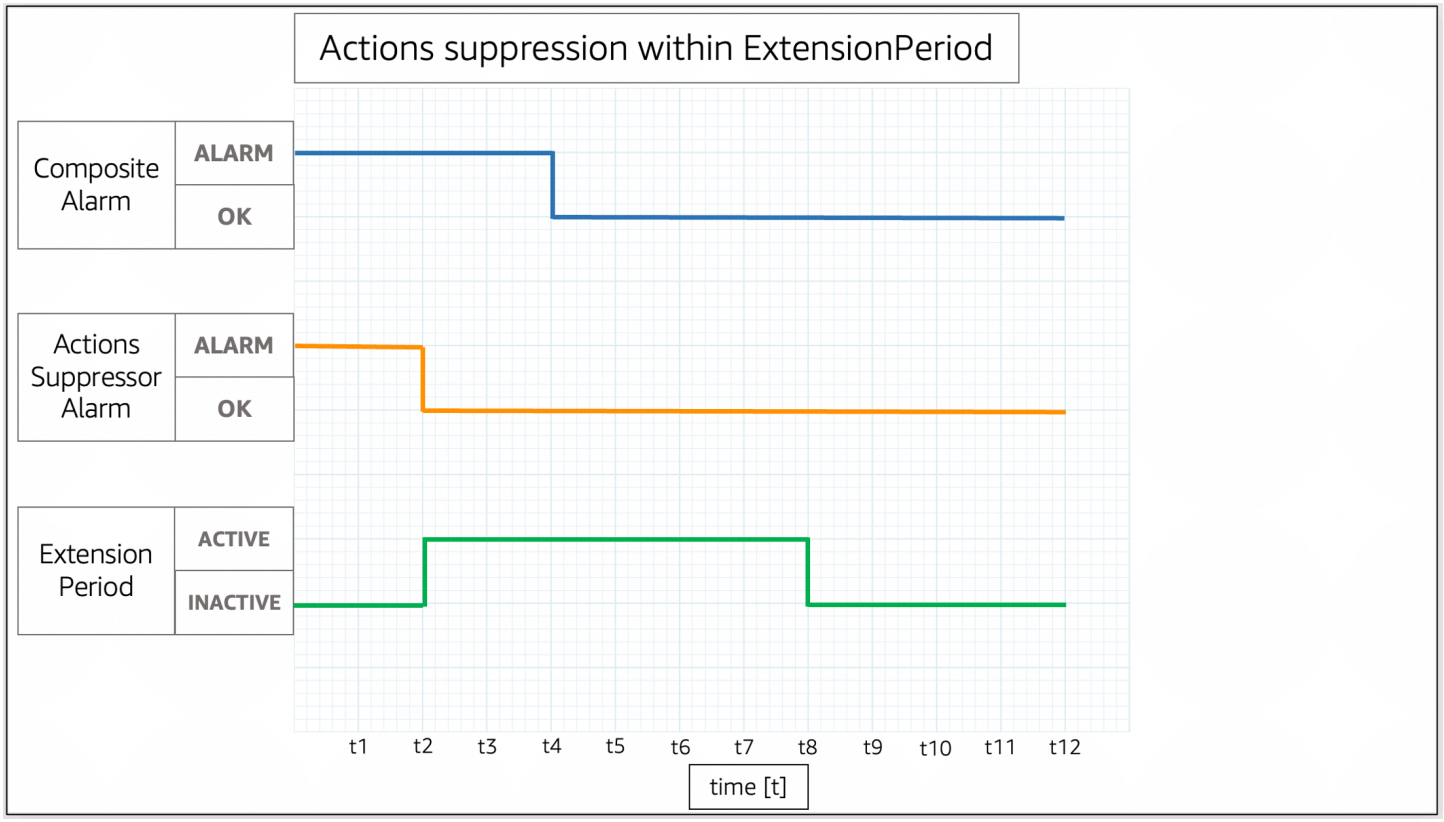
`WaitPeriod` および `ExtensionPeriod`

サプレッサーアラームを指定する際は、WaitPeriod および ExtensionPeriod パラメータを設定します。これらのパラメータにより、サプレッサーアラームの状態の変更中に、複合アラームにより予期しないアクションが実行されるのを防ぐことができます。WaitPeriod を使用すると、サプレッサーアラームが OK から ALARM に変わる際に発生する可能性のある遅延を補うことができます。例えば、サプレッサーアラームが 60 秒以内に OK から ALARM に変わる場合、WaitPeriod を 60 秒に設定します。



この図では、複合アラームは t2 で OK から ALARM に変わります。WaitPeriod は t2 で開始され、t8 で終了します。これにより、WaitPeriod が t8 で期限切れになる際、複合アラームのアクションを抑制する前に、t4 でサプレッサーアラームの状態が OK から ALARM に変わる時間を確保できます。

ExtensionPeriod を使用すると、サプレッサーアラームが OK に変わった後、複合アラームが OK に変わる際に発生する可能性のある遅延を補うことができます。例えば、サプレッサーアラームが OK に変わってから 60 秒以内に複合アラームが OK に変わる場合、ExtensionPeriod を 60 秒に設定します。



この図では、サプレッサーアラームは t2 で ALARM から OK に変わります。ExtensionPeriod は t2 で開始され、t8 で終了します。これにより、ExtensionPeriod が t8 で期限切れになる前に、複合アラームが ALARM から OK に変わる時間を確保できます。

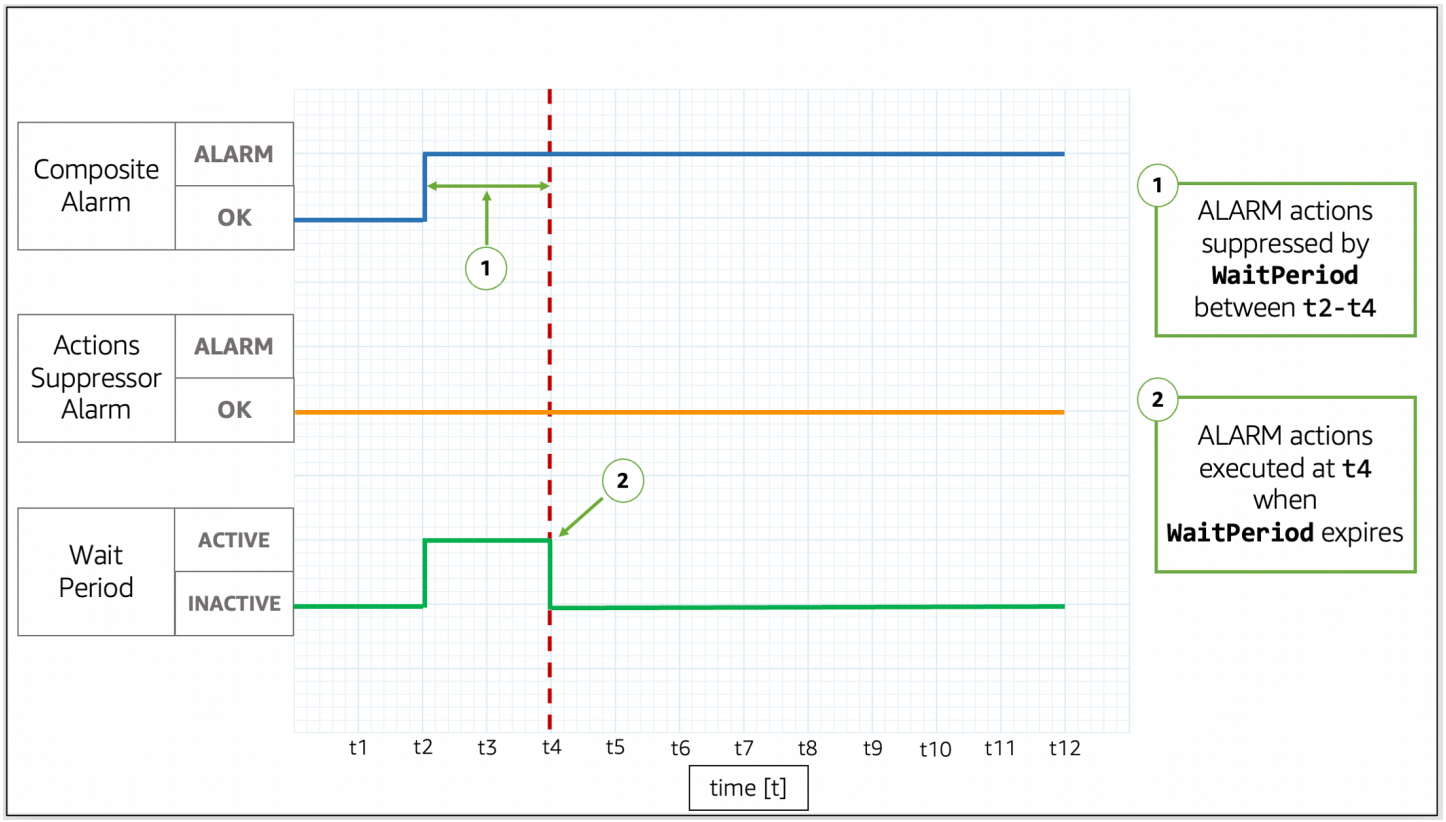
WaitPeriod および ExtensionPeriod がアクティブになるとき、複合アラームではアクションは実行されません。ExtensionPeriod および WaitPeriod が非アクティブになるとき、複合アラームでは現在の状態に基づいてアクションが実行されます。CloudWatch では 1 分ごとにメトリクスアラームが評価されるため、各パラメータの値は 60 秒に設定することをお勧めします。パラメータは、秒単位で任意の整数に設定できます。

次の例は、WaitPeriod および ExtensionPeriod を使用して、複合アラームで予期しないアクションが実行されるのを防ぐ方法を、より詳細に説明しています。

> **Note**
>
> 次の例では、WaitPeriod は 2 つの時間単位、ExtensionPeriod は 3 つの時間単位として設定されています。

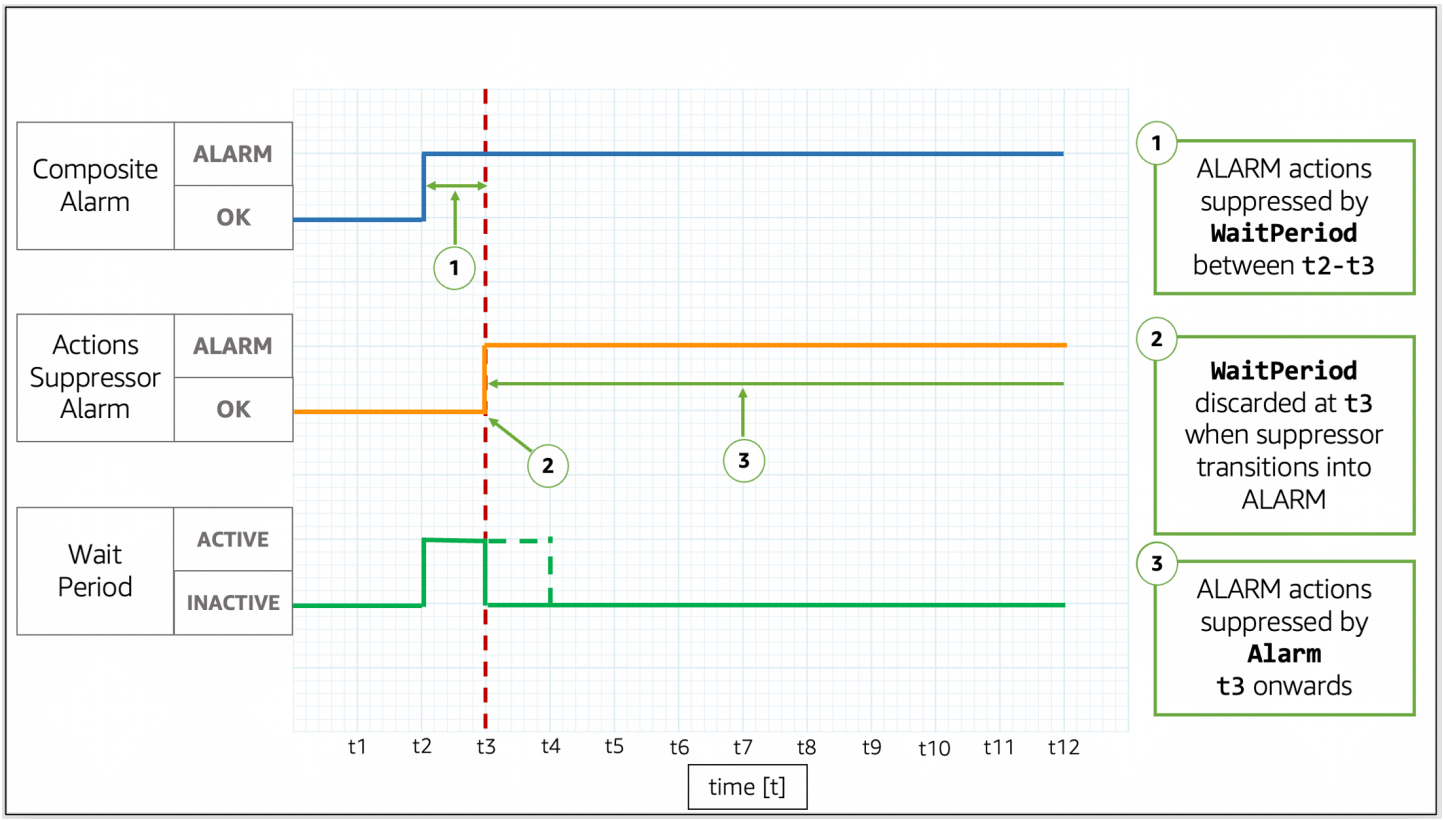
例

例 1: **WaitPeriod** の後にアクションが抑制されない



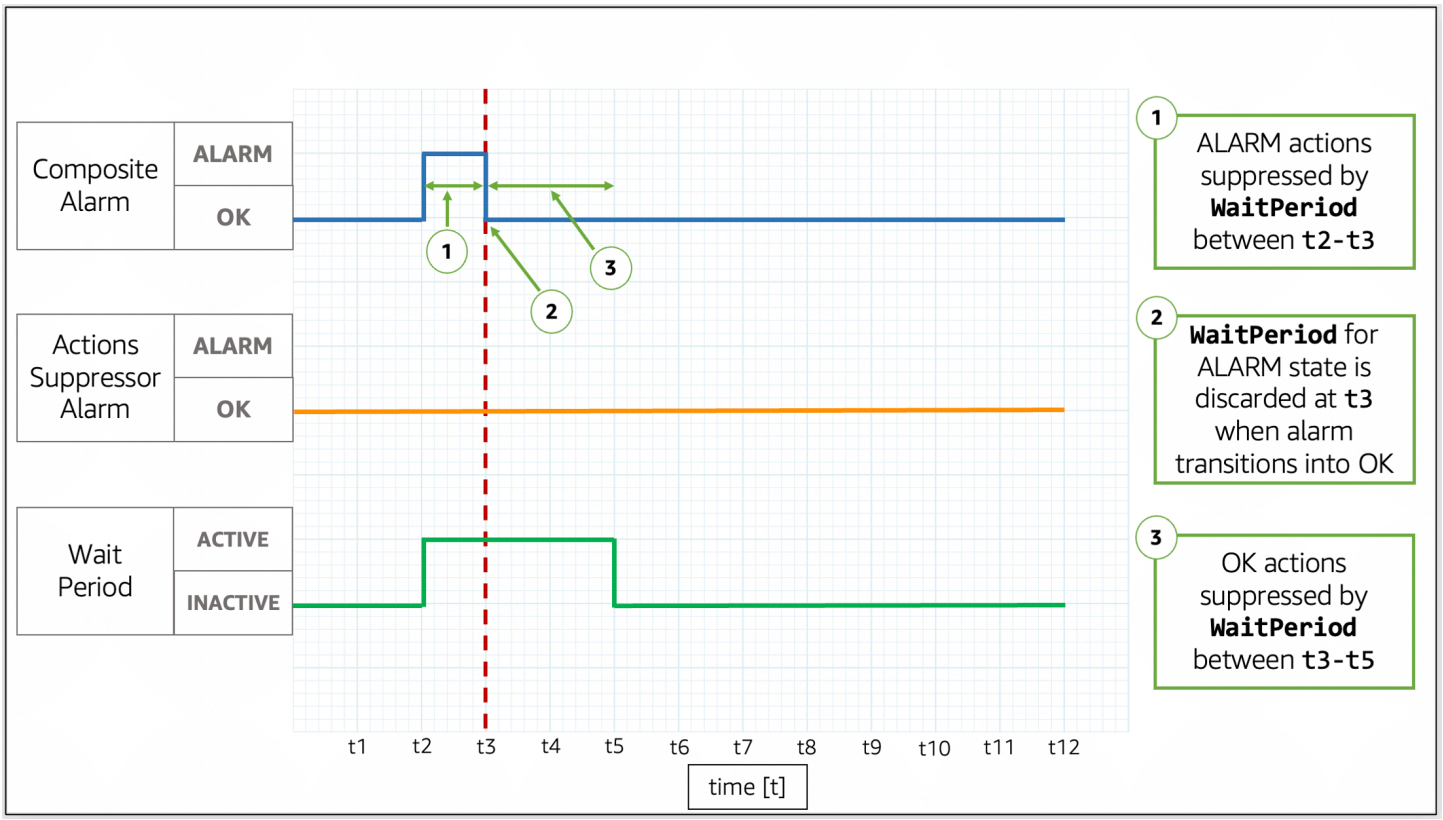
この図では、複合アラームの状態が t2 で OK から ALARM に変わっています。WaitPeriod は t2 で開始され t4 で終了するため、複合アラームでアクションが実行されるのを防ぐことができます。t4 で WaitPeriod の期限が切れた後は、サプレッサーアラームがまだ OK 状態であるため、複合アラームでアクションが実行されます。

例 2: **WaitPeriod** の期限が切れる前にアラームによりアクションが抑制される



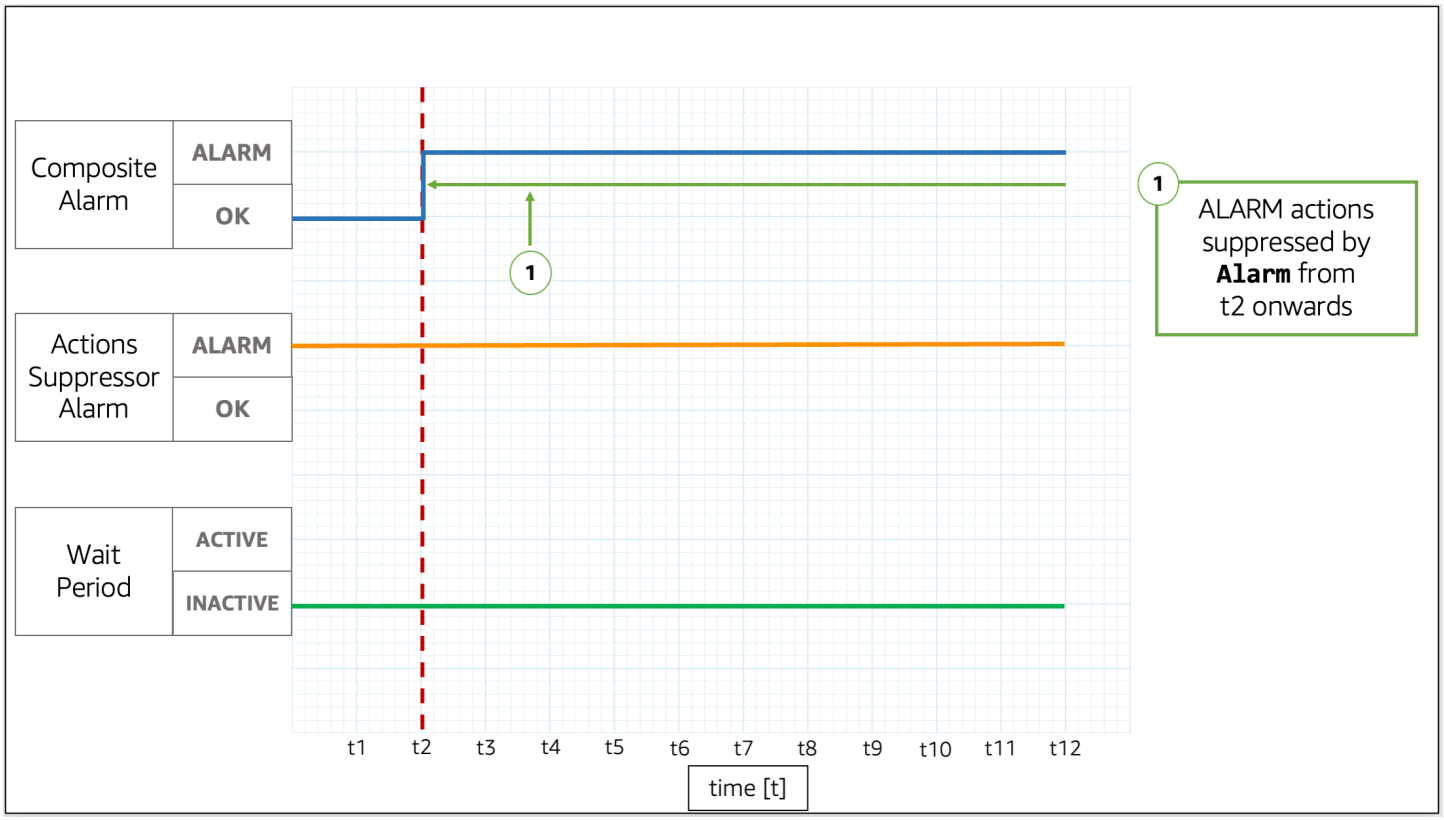
この図では、複合アラームの状態が t2 で OK から ALARM に変わっています。WaitPeriod は t2 で開始され、t4 で終了します。これにより、t3 でサプレッサーアラームの状態が OK から ALARM に変わる時間を確保できます。サプレッサーアラームの状態が t3 で OK から ALARM に変わるため、t2 で開始された WaitPeriod は破棄されます。また、サプレッサーアラームにより複合アラームのアクションの実行が停止されます。

例 3: アクションが **WaitPeriod** により抑制される場合の状態遷移



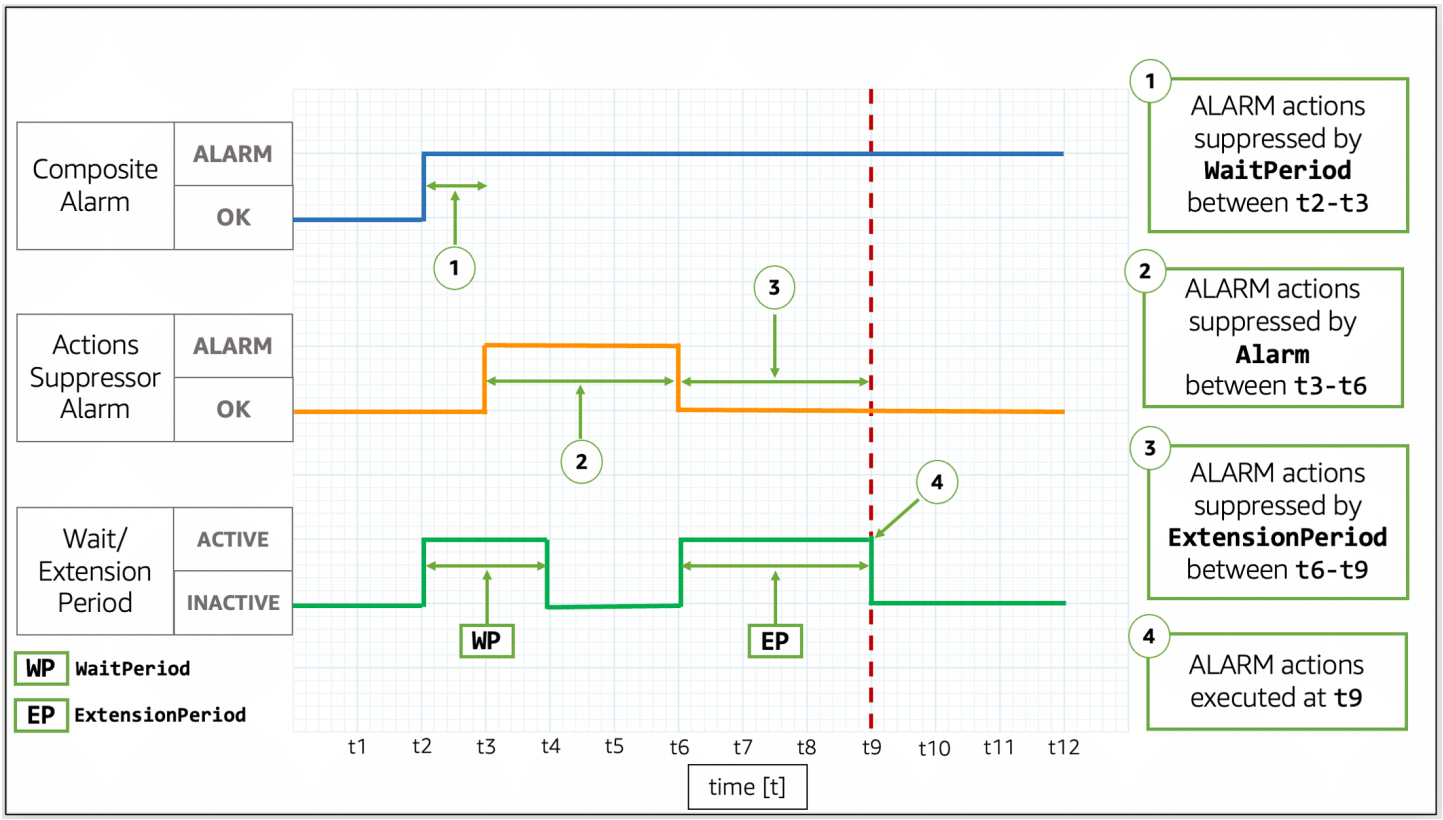
この図では、複合アラームの状態が t2 で OK から ALARM に変わっています。WaitPeriod は t2 で開始され、t4 で終了します。これにより、サプレッサーアラームの状態が変わる時間を確保できます。複合アラームは t3 でOK に戻るため、t2 で開始された WaitPeriod は破棄されます。新たに t3 で WaitPeriod が開始され、t5 で終了します。t5 で新しい WaitPeriod の期限が切れると、複合アラームによりアクションが実行されます。

例 4: アクションがアラームにより抑制される場合の状態遷移



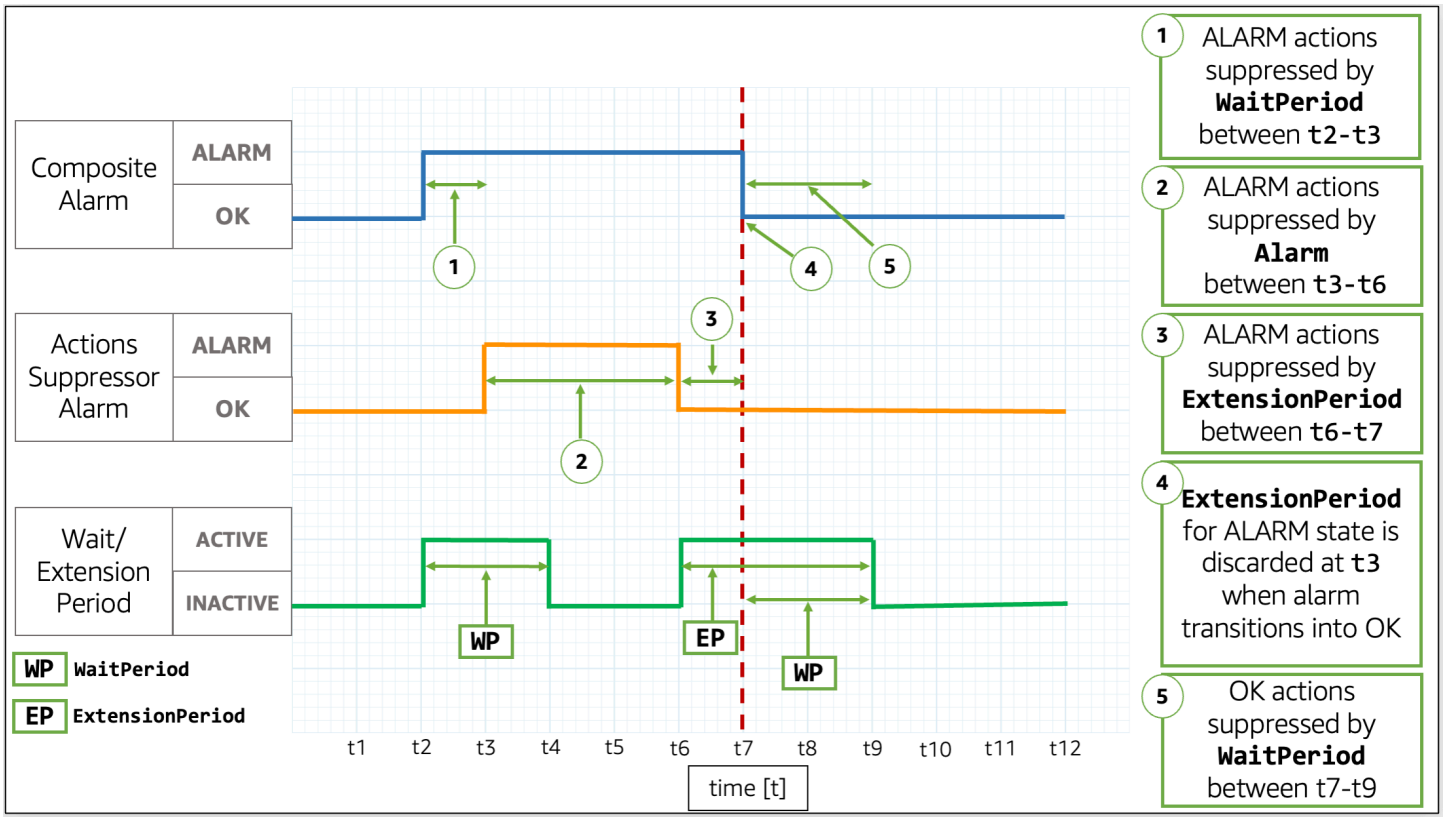
この図では、複合アラームの状態が t2 で OK から ALARM に変わっています。サプレッサーアラームは既に ALARM の状態です。サプレッサーアラームにより、複合アラームによるアクションの実行が停止されます。

例 5: **ExtensionPeriod** の後にアクションが抑制されない



この図では、複合アラームの状態が t2 で OK から ALARM に変わっています。WaitPeriod は t2 で開始され、t4 で終了します。これにより、複合アラームのアクションを t6 まで抑制する前に、t3 でサプレッサーアラームの状態が OK から ALARM に変わる時間を確保できます。サプレッサーアラームの状態が t3 で OK から ALARM に変わるため、t2 で開始された WaitPeriod は破棄されます。t6 で、サプレッサーアラームは OK に変わります。ExtensionPeriod は t6 で開始され、t9 で終了します。ExtensionPeriod の期限が切れると、複合アラームによりアクションが実行されます。

例 6: アクションが**ExtensionPeriod** により抑制される場合の状態遷移



この図では、複合アラームの状態が t2 で OK から ALARM に変わっています。WaitPeriod は t2 で開始され、t4 で終了します。これにより、複合アラームのアクションを t6 まで抑制する前に、t3 でサプレッサーアラームの状態が OK から ALARM に変わる時間を確保できます。サプレッサーアラームの状態が t3 で OK から ALARM に変わるため、t2 で開始された WaitPeriod は破棄されます。t6 で、サプレッサーアラームは OK に戻ります。ExtensionPeriod は t6 で開始され、t9 で終了します。t7 で複合アラームが OK に戻ると、ExtensionPeriod は破棄されます。その後 t7 では新しい WaitPeriod が開始され、t9 で終了します。

> **Tip**
>
> アクションのサプレッサーアラームを置き換えると、アクティブな WaitPeriod または ExtensionPeriod は破棄されます。

アラームの変更への対応

CloudWatch は、2 種類のアラーム変更についてユーザーに通知できます。1 つはアラームの状態を変更したときで、もう 1 つはアラームの設定が更新されたときです。

アラームが評価されると、「ALARM」、「OK」、「INSUFFICIENT_DATA」など、ある状態から別の状態に変化することがあります。このようなアラーム状態の変化により、インシデントが発生する可能性や、通常の状態への復帰、メトリクスが利用不可であることを知らせることができます。このような場合は、以下のオプションのいずれかを使用して、ユーザーへ働きかけたり通知したりすることをお勧めします。

- アラームのアクションの一環として、SNS トピックに通知を送信するようにアラームを設定できます。SNS トピックは、Application-to-Application (A2A) メッセージングと、E メール通知や SMS などのチャネルを含む Application-to-Person (A2P) 通知用に設定できます。SNS トピック用に定義したすべての宛先がアラーム通知を受信します。詳細については、「Amazon SNS イベントの送信先」を参照してください。
- アラーム状態変更イベントの通知を設定できます。 AWSユーザー通知には、このような通知を設定するに設定すネイティブの方法があり、これが推奨される方法です。

さらに、アラームの状態が変化した時や、アラームが作成、削除、または更新された時に、CloudWatch は Amazon EventBridge にイベントを送信します。EventBridge ルールを記述してアクションを実行したり、EventBridge がこれらのイベントを受信したときに通知を受け取ることができます。

トピック

- アラームの変更をユーザーに通知する
- アラームイベントと EventBridge

アラームの変更をユーザーに通知する

このセクションでは、AWS ユーザー通知または Amazon Simple Notification Service を使用して、ユーザーにアラームの変更を通知する方法を説明します。

AWS ユーザー通知を設定する

AWS User Notifications を使用して、CloudWatch アラーム状態変更、設定変更イベントに関する通知を送信する配信チャネルを設定できます。指定したルールにイベントが一致すると、通知を受け取ります。イベントの通知は、E メール、AWS Chatbot のチャット通知、AWS コンソールモバイルアプリケーションのプッシュ通知 など、複数のチャネルで受け取ることができます。[コンソール通知センター] で通知を確認することもできます。ユーザー通知は集約をサポートしているため、特定のイベント中に受け取る通知の数を減らすことができます。

AWS User Notifications で作成した通知設定は、ターゲットアラーム状態ごとに設定できるアクション数の制限にはカウントされません。AWS User Notifications は Amazon EventBridge に送信されるイベントと一致すると、特定のアラームまたはパターンの許可リストまたは拒否リストの高度なフィルターを指定しない限り、アカウントと選択したリージョンのすべてのアラームに対する通知を送信します。

以下の高度なフィルタの例では、ServerCpuTooHigh という名前のアラームのアラーム状態が OK から ALARM に変化したことを照合します。

```
{
"detail": {
    "alarmName": ["ServerCpuTooHigh"],
    "previousState": { "value": ["OK"] },
    "state": { "value": ["ALARM"] }
  }
}
```

任意の EventBridge イベントのアラームによって公開されるプロパティを使用してフィルターを作成できます。詳細については、「アラームイベントと EventBridge」を参照してください。

Amazon SNS 通知の設定

Amazon Simple Notification Service を使用して、モバイルテキストメッセージング (SMS) や E メールメッセージなどのアプリケーション間 (A2A) メッセージングと、アプリケーション対個人間 (A2P) メッセージングの両方を送信できます。詳細については、「Amazon SNS イベントの送信先」を参照してください。

アラームが取りうる状態ごとに、SNS トピックにメッセージを送信するようにアラームを設定できます。特定のアラームの状態に設定するすべての Amazon SNS トピックは、そのアラームと状態に設定できるアクション数の制限にカウントされます。アカウント内のどのアラームからでも同じ Amazon SNS トピックにメッセージを送信でき、アプリケーション間 (A2A) とアプリケーション対個人(A2P) 両方のコンシューマーに同じ Amazon SNS トピックを使用できます。この設定はアラームレベルで行われるため、設定したアラームのみが選択した Amazon SNS トピックにメッセージを送信します。

最初にトピックを作成して、それをサブスクライブします。オプションでトピックにテストメッセージを発行できます。例については、「 を使用した Amazon SNS トピックの設定AWS Management Console」を参照してください。または、詳細については、「Amazon SNS の始め方」を参照してください。

または、AWS Management Console を使用して CloudWatch アラームを作成する予定である場合は、アラームの作成時にトピックを作成できるため、この手順を省略できます。

CloudWatch アラームを作成すると、アラームが入る任意のターゲット状態に対してアクションを追加できます。通知を受け取りたい状態の Amazon SNS 通知を追加し、前のステップで作成した Amazon SNS トピックを選択して、アラームが選択した状態になったときに E メール通知を送信します。

> **Note**
>
> Amazon SNS トピックを作成する際は、スタンダードトピックまたは FIFO トピックを選択します。CloudWatch では、両方のタイプのトピックへのアラーム通知のすべてが確実に公開されます。ただし、FIFO トピックを使用した場合でも、まれに CloudWatch により順不同でトピックに通知が送信されることがあります。FIFO トピックを使用する場合、アラームはアラーム通知のメッセージグループ ID がアラームの ARN のハッシュになるように設定されます。

混乱した代理問題の防止

クロスサービスの混乱した代理のセキュリティ上の問題を防ぐために、CloudWatch に Amazon SNS リソースへのアクセスを許可する Amazon SNS リソースポリシーで `aws:SourceArn` および `aws:SourceAccount` グローバルコンディションキーを使用することをお勧めします。

次のリソースポリシーの例では、`aws:SourceArn` 条件キーを使用して、指定されたアカウントの CloudWatch アラームでのみ使用される `SNS:Publish` 許可を絞り込みます。

```
{
    "Statement": [{
        "Effect": "Allow",
        "Principal": {
            "Service": "cloudwatch.amazonaws.com"
        },
        "Action": "SNS:Publish",
        "Resource": "arn:aws:sns:us-east-2:444455556666:MyTopic",
        "Condition": {
            "ArnLike": {
                "aws:SourceArn": "arn:aws:cloudwatch:us-east-2:111122223333:alarm:*"
            },
            "StringEquals": {
                "aws:SourceAccount": "111122223333"
```

```
            }
        }
    }]
}
```

アラーム ARN に非 ASCII 文字が含まれている場合は、`aws:SourceAccount` グローバル条件キーのみを使用して、許可を制限してください。

を使用した Amazon SNS トピックの設定AWS Management Console

最初にトピックを作成して、それをサブスクライブします。オプションでトピックにテストメッセージを発行できます。

SNS トピックを作成するには

1. Amazon SNS コンソール (https://console.aws.amazon.com/sns/v3/home) を開きます。
2. Amazon SNS ダッシュボードの [Common actions] の下で、[Create topic] を選択します。
3. [新しいトピックの作成] ダイアログボックスで、[トピック] にトピックの名前 (例: **`my-topic`**) を入力します。
4. [Create topic] (トピックの作成) を選択します。
5. 次のタスクの [Topic ARN] (例: arn:aws:sns:us-east-1:111122223333:my-topic) をコピーします。

SNS トピックをサブスクライブするには

1. Amazon SNS コンソール (https://console.aws.amazon.com/sns/v3/home) を開きます。
2. ナビゲーションペインで、[サブスクリプション]、[サブスクリプションの作成] の順に選択します。
3. [サブスクリプションの作成] ダイアログボックスの [トピック ARN] に、前のタスクで作成したトピック ARN を貼り付けます。
4. [Protocol (プロトコル)] として [Email (E メール)] を選択します。
5. [エンドポイント] に通知の受取先として使用する E メールアドレスを入力し、[サブスクリプションの作成] を選択します。
6. E メールアプリケーションで AWS 通知から届いたメッセージを開き、サブスクライブを確認します。

   ウェブブラウザに Amazon SNS の確認画面が表示されます。

SNS トピックにテストメッセージを発行するには

1. Amazon SNS コンソール (https://console.aws.amazon.com/sns/v3/home) を開きます。
2. ナビゲーションペインで、[トピック] を選択します。
3. [トピック] ページでトピックを選択し、[トピックに発行] を選択します。
4. [メッセージを発行] ページの [件名] にメッセージの件名を入力し、[メッセージ] に短いメッセージを入力します。
5. [メッセージの発行] を選択します。
6. メールをチェックして、メッセージを受信したことを確認します。

を使用した SNS トピックの設定AWS CLI

最初に SNS トピックを作成し、次にメッセージをトピックに直接発行して設定が正しいことをテストします。

SNS トピックを設定するには

1. 次のように create-topic コマンドを使用してトピックを作成します。

```
aws sns create-topic --name my-topic
```

Amazon SNS は次の形式でトピック ARN を返します。

```
{
    "TopicArn": "arn:aws:sns:us-east-1:111122223333:my-topic"
}
```

2. subscribe コマンドを用いて、自分の E メールアドレスをトピックにサブスクライブします。サブスクライブのリクエストが成功すると、確認メールが届きます。

```
aws sns subscribe --topic-arn arn:aws:sns:us-east-1:111122223333:my-topic --
protocol email --notification-endpoint my-email-address
```

Amazon SNS は以下を返します。

```
{
    "SubscriptionArn": "pending confirmation"
}
```

3. E メールアプリケーションで AWS 通知から届いたメッセージを開き、サブスクライブを確認します。

   ウェブブラウザに、Amazon Simple Notification Service からの確認応答が表示されます。

4. list-subscriptions-by-topic コマンドを使用して、サブスクリプションを確認します。

```
aws sns list-subscriptions-by-topic --topic-arn arn:aws:sns:us-
east-1:111122223333:my-topic
```

   Amazon SNS は以下を返します。

```
{
  "Subscriptions": [
    {
        "Owner": "111122223333",
        "Endpoint": "me@mycompany.com",
        "Protocol": "email",
        "TopicArn": "arn:aws:sns:us-east-1:111122223333:my-topic",
        "SubscriptionArn": "arn:aws:sns:us-east-1:111122223333:my-topic:64886986-
bf10-48fb-a2f1-dab033aa67a3"
    }
  ]
}
```

5. (オプション) publish コマンドを使用してトピックにテストメッセージを発行します。

```
aws sns publish --message "Verification" --topic arn:aws:sns:us-
east-1:111122223333:my-topic
```

   Amazon SNS は以下を返します。

```
{
    "MessageId": "42f189a0-3094-5cf6-8fd7-c2dde61a4d7d"
}
```

6. メールをチェックして、メッセージを受信したことを確認します。

アラームイベントと EventBridge

CloudWatch アラームが作成、更新、削除されたり、アラームの状態が変更されるたびに、CloudWatch は Amazon EventBridge にイベントを送信します。EventBridge およびこれらのイベントを使用して、アラーム状態が変わったときに通知するなどのアクションを実行するルールを記述できます。詳細については、「Amazon EventBridge とは」を参照してください。

CloudWatch は、アラーム状態変更イベントが EventBridge に確実に配信されるようにします。

CloudWatch のサンプルイベント

このセクションでは、CloudWatch からのイベント例を示します。

単一のメトリクスに基づくアラームの状態変化

```
{
    "version": "0",
    "id": "c4c1c1c9-6542-e61b-6ef0-8c4d36933a92",
    "detail-type": "CloudWatch Alarm State Change",
    "source": "aws.cloudwatch",
    "account": "123456789012",
    "time": "2019-10-02T17:04:40Z",
    "region": "us-east-1",
    "resources": [
        "arn:aws:cloudwatch:us-east-1:123456789012:alarm:ServerCpuTooHigh"
    ],
    "detail": {
        "alarmName": "ServerCpuTooHigh",
        "configuration": {
            "description": "Goes into alarm when server CPU utilization is too high!",
            "metrics": [
                {
                    "id": "30b6c6b2-a864-43a2-4877-c09a1afc3b87",
                    "metricStat": {
                        "metric": {
                            "dimensions": {
                                "InstanceId": "i-12345678901234567"
                            },
                            "name": "CPUUtilization",
                            "namespace": "AWS/EC2"
                        },
                        "period": 300,
                        "stat": "Average"
```

```
                    },
                    "returnData": true
                }
            ]
        },
        "previousState": {
            "reason": "Threshold Crossed: 1 out of the last 1 datapoints
 [0.0666851903306472 (01/10/19 13:46:00)] was not greater than the threshold (50.0)
 (minimum 1 datapoint for ALARM -> OK transition).",
            "reasonData": "{\"version\":\"1.0\",\"queryDate\":
\"2019-10-01T13:56:40.985+0000\",\"startDate\":\"2019-10-01T13:46:00.000+0000\",
\"statistic\":\"Average\",\"period\":300,\"recentDatapoints\":[0.0666851903306472],
\"threshold\":50.0}",
            "timestamp": "2019-10-01T13:56:40.987+0000",
            "value": "OK"
        },
        "state": {
            "reason": "Threshold Crossed: 1 out of the last 1 datapoints
 [99.50160229693434 (02/10/19 16:59:00)] was greater than the threshold (50.0) (minimum
 1 datapoint for OK -> ALARM transition).",
            "reasonData": "{\"version\":\"1.0\",\"queryDate\":
\"2019-10-02T17:04:40.985+0000\",\"startDate\":\"2019-10-02T16:59:00.000+0000\",
\"statistic\":\"Average\",\"period\":300,\"recentDatapoints\":[99.50160229693434],
\"threshold\":50.0}",
            "timestamp": "2019-10-02T17:04:40.989+0000",
            "value": "ALARM"
        }
    }
}
```

メトリクスの数式に基づくアラームの状態変化

```
{
    "version": "0",
    "id": "2dde0eb1-528b-d2d5-9ca6-6d590caf2329",
    "detail-type": "CloudWatch Alarm State Change",
    "source": "aws.cloudwatch",
    "account": "123456789012",
    "time": "2019-10-02T17:20:48Z",
    "region": "us-east-1",
    "resources": [
        "arn:aws:cloudwatch:us-east-1:123456789012:alarm:TotalNetworkTrafficTooHigh"
    ],
```

```
    "detail": {
        "alarmName": "TotalNetworkTrafficTooHigh",
        "configuration": {
            "description": "Goes into alarm if total network traffic exceeds 10Kb",
            "metrics": [
                {
                    "expression": "SUM(METRICS())",
                    "id": "e1",
                    "label": "Total Network Traffic",
                    "returnData": true
                },
                {
                    "id": "m1",
                    "metricStat": {
                        "metric": {
                            "dimensions": {
                                "InstanceId": "i-12345678901234567"
                            },
                            "name": "NetworkIn",
                            "namespace": "AWS/EC2"
                        },
                        "period": 300,
                        "stat": "Maximum"
                    },
                    "returnData": false
                },
                {
                    "id": "m2",
                    "metricStat": {
                        "metric": {
                            "dimensions": {
                                "InstanceId": "i-12345678901234567"
                            },
                            "name": "NetworkOut",
                            "namespace": "AWS/EC2"
                        },
                        "period": 300,
                        "stat": "Maximum"
                    },
                    "returnData": false
                }
            ]
        },
        "previousState": {
```

```
            "reason": "Unchecked: Initial alarm creation",
            "timestamp": "2019-10-02T17:20:03.642+0000",
            "value": "INSUFFICIENT_DATA"
        },
        "state": {
            "reason": "Threshold Crossed: 1 out of the last 1 datapoints [45628.0
 (02/10/19 17:10:00)] was greater than the threshold (10000.0) (minimum 1 datapoint for
 OK -> ALARM transition).",
            "reasonData": "{\"version\":\"1.0\",\"queryDate\":
\"2019-10-02T17:20:48.551+0000\",\"startDate\":\"2019-10-02T17:10:00.000+0000\",
\"period\":300,\"recentDatapoints\":[45628.0],\"threshold\":10000.0}",
            "timestamp": "2019-10-02T17:20:48.554+0000",
            "value": "ALARM"
        }
    }
}
```

異常検出アラームの状態変化

```
{
    "version": "0",
    "id": "daafc9f1-bddd-c6c9-83af-74971fcfc4ef",
    "detail-type": "CloudWatch Alarm State Change",
    "source": "aws.cloudwatch",
    "account": "123456789012",
    "time": "2019-10-03T16:00:04Z",
    "region": "us-east-1",
    "resources": ["arn:aws:cloudwatch:us-east-1:123456789012:alarm:EC2 CPU Utilization
 Anomaly"],
    "detail": {
        "alarmName": "EC2 CPU Utilization Anomaly",
        "state": {
            "value": "ALARM",
            "reason": "Thresholds Crossed: 1 out of the last 1 datapoints [0.0
 (03/10/19 15:58:00)] was less than the lower thresholds [0.020599444741798756] or
 greater than the upper thresholds [0.3006915352732461] (minimum 1 datapoint for OK ->
 ALARM transition).",
            "reasonData": "{\"version\":\"1.0\",\"queryDate\":
\"2019-10-03T16:00:04.650+0000\",\"startDate\":\"2019-10-03T15:58:00.000+0000\",
\"period\":60,\"recentDatapoints\":[0.0],\"recentLowerThresholds\":
[0.020599444741798756],\"recentUpperThresholds\":[0.3006915352732461]}",
            "timestamp": "2019-10-03T16:00:04.653+0000"
        },
```

```
        "previousState": {
            "value": "OK",
            "reason": "Thresholds Crossed: 1 out of the last 1 datapoints
 [0.166666666664241 (03/10/19 15:57:00)] was not less than the lower thresholds
 [0.0206719426210418] or not greater than the upper thresholds [0.30076870222143803]
 (minimum 1 datapoint for ALARM -> OK transition).",
            "reasonData": "{\"version\":\"1.0\",\"queryDate\":
\"2019-10-03T15:59:04.670+0000\",\"startDate\":\"2019-10-03T15:57:00.000+0000\",
\"period\":60,\"recentDatapoints\":[0.166666666664241],\"recentLowerThresholds\":
[0.0206719426210418],\"recentUpperThresholds\":[0.30076870222143803]}",
            "timestamp": "2019-10-03T15:59:04.672+0000"
        },
        "configuration": {
            "description": "Goes into alarm if CPU Utilization is out of band",
            "metrics": [{
                "id": "m1",
                "metricStat": {
                    "metric": {
                        "namespace": "AWS/EC2",
                        "name": "CPUUtilization",
                        "dimensions": {
                            "InstanceId": "i-12345678901234567"
                        }
                    },
                    "period": 60,
                    "stat": "Average"
                },
                "returnData": true
            }, {
                "id": "ad1",
                "expression": "ANOMALY_DETECTION_BAND(m1, 0.8)",
                "label": "CPUUtilization (expected)",
                "returnData": true
            }]
        }
    }
}
```

サプレッサーアラーム付き複合アラームの状態変化

```
{
    "version": "0",
    "id": "d3dfc86d-384d-24c8-0345-9f7986db0b80",
```

```
    "detail-type": "CloudWatch Alarm State Change",
    "source": "aws.cloudwatch",
    "account": "123456789012",
    "time": "2022-07-22T15:57:45Z",
    "region": "us-east-1",
    "resources": [
        "arn:aws:cloudwatch:us-east-1:123456789012:alarm:ServiceAggregatedAlarm"
    ],
    "detail": {
        "alarmName": "ServiceAggregatedAlarm",
        "state": {
            "actionsSuppressedBy": "WaitPeriod",
            "actionsSuppressedReason": "Actions suppressed by WaitPeriod",
            "value": "ALARM",
            "reason": "arn:aws:cloudwatch:us-
east-1:123456789012:alarm:SuppressionDemo.EventBridge.FirstChild transitioned to ALARM
 at Friday 22 July, 2022 15:57:45 UTC",
            "reasonData": "{\"triggeringAlarms\":[{\"arn\":\"arn:aws:cloudwatch:us-
east-1:123456789012:alarm:ServerCpuTooHigh\",\"state\":{\"value\":\"ALARM\",\"timestamp
\":\"2022-07-22T15:57:45.394+0000\"}}]}",
            "timestamp": "2022-07-22T15:57:45.394+0000"
        },
        "previousState": {
            "value": "OK",
            "reason": "arn:aws:cloudwatch:us-
east-1:123456789012:alarm:SuppressionDemo.EventBridge.Main was created and its alarm
 rule evaluates to OK",
            "reasonData": "{\"triggeringAlarms\":[{\"arn\":\"arn:aws:cloudwatch:us-
east-1:123456789012:alarm:TotalNetworkTrafficTooHigh\",\"state\":{\"value\":\"OK\",
\"timestamp\":\"2022-07-14T16:28:57.770+0000\"}},{\"arn\":\"arn:aws:cloudwatch:us-
east-1:123456789012:alarm:ServerCpuTooHigh\",\"state\":{\"value\":\"OK\",\"timestamp\":
\"2022-07-14T16:28:54.191+0000\"}}]}",
            "timestamp": "2022-07-22T15:56:14.552+0000"
        },
        "configuration": {
            "alarmRule": "ALARM(ServerCpuTooHigh) OR
 ALARM(TotalNetworkTrafficTooHigh)",
            "actionsSuppressor": "ServiceMaintenanceAlarm",
            "actionsSuppressorWaitPeriod": 120,
            "actionsSuppressorExtensionPeriod": 180
        }
    }
}
```

複合アラームの作成


```
{

    "version": "0",
    "id": "91535fdd-1e9c-849d-624b-9a9f2b1d09d0",
    "detail-type": "CloudWatch Alarm Configuration Change",
    "source": "aws.cloudwatch",
    "account": "123456789012",
    "time": "2022-03-03T17:06:22Z",
    "region": "us-east-1",
    "resources": [
        "arn:aws:cloudwatch:us-east-1:123456789012:alarm:ServiceAggregatedAlarm"
    ],
    "detail": {
        "alarmName": "ServiceAggregatedAlarm",
        "operation": "create",
        "state": {
            "value": "INSUFFICIENT_DATA",
            "timestamp": "2022-03-03T17:06:22.289+0000"
        },
        "configuration": {
            "alarmRule": "ALARM(ServerCpuTooHigh) OR
 ALARM(TotalNetworkTrafficTooHigh)",
            "alarmName": "ServiceAggregatedAlarm",
            "description": "Aggregated monitor for instance",
            "actionsEnabled": true,
            "timestamp": "2022-03-03T17:06:22.289+0000",
            "okActions": [],
            "alarmActions": [],
            "insufficientDataActions": []
        }
    }
}
```


サプレッサーアラーム付き複合アラームの作成


```
{
    "version": "0",
    "id": "454773e1-09f7-945b-aa2c-590af1c3f8e0",
    "detail-type": "CloudWatch Alarm Configuration Change",
    "source": "aws.cloudwatch",
    "account": "123456789012",
```

```
    "time": "2022-07-14T13:59:46Z",
    "region": "us-east-1",
    "resources": [
        "arn:aws:cloudwatch:us-east-1:123456789012:alarm:ServiceAggregatedAlarm"
    ],
    "detail": {
        "alarmName": "ServiceAggregatedAlarm",
        "operation": "create",
        "state": {
            "value": "INSUFFICIENT_DATA",
            "timestamp": "2022-07-14T13:59:46.425+0000"
        },
        "configuration": {
            "alarmRule": "ALARM(ServerCpuTooHigh) OR
 ALARM(TotalNetworkTrafficTooHigh)",
            "actionsSuppressor": "ServiceMaintenanceAlarm",
            "actionsSuppressorWaitPeriod": 120,
            "actionsSuppressorExtensionPeriod": 180,
            "alarmName": "ServiceAggregatedAlarm",
            "actionsEnabled": true,
            "timestamp": "2022-07-14T13:59:46.425+0000",
            "okActions": [],
            "alarmActions": [],
            "insufficientDataActions": []
        }
    }
}
```

メトリクスアラームの更新

```
{

    "version": "0",
    "id": "bc7d3391-47f8-ae47-f457-1b4d06118d50",
    "detail-type": "CloudWatch Alarm Configuration Change",
    "source": "aws.cloudwatch",
    "account": "123456789012",
    "time": "2022-03-03T17:06:34Z",
    "region": "us-east-1",
    "resources": [
        "arn:aws:cloudwatch:us-east-1:123456789012:alarm:ServerCpuTooHigh"
    ],
    "detail": {
```

```
        "alarmName": "ServerCpuTooHigh",
        "operation": "update",
        "state": {
            "value": "INSUFFICIENT_DATA",
            "timestamp": "2022-03-03T17:06:13.757+0000"
        },
        "configuration": {
            "evaluationPeriods": 1,
            "threshold": 80,
            "comparisonOperator": "GreaterThanThreshold",
            "treatMissingData": "ignore",
            "metrics": [
                {
                    "id": "86bfa85f-b14c-ebf7-8916-7da014ce23c0",
                    "metricStat": {
                        "metric": {
                            "namespace": "AWS/EC2",
                            "name": "CPUUtilization",
                            "dimensions": {
                                "InstanceId": "i-12345678901234567"
                            }
                        },
                        "period": 300,
                        "stat": "Average"
                    },
                    "returnData": true
                }
            ],
            "alarmName": "ServerCpuTooHigh",
            "description": "Goes into alarm when server CPU utilization is too high!",
            "actionsEnabled": true,
            "timestamp": "2022-03-03T17:06:34.267+0000",
            "okActions": [],
            "alarmActions": [],
            "insufficientDataActions": []
        },
        "previousConfiguration": {
            "evaluationPeriods": 1,
            "threshold": 70,
            "comparisonOperator": "GreaterThanThreshold",
            "treatMissingData": "ignore",
            "metrics": [
                {
                    "id": "d6bfa85f-893e-b052-a58b-4f9295c9111a",
```

```
                    "metricStat": {
                        "metric": {
                            "namespace": "AWS/EC2",
                            "name": "CPUUtilization",
                            "dimensions": {
                                "InstanceId": "i-12345678901234567"
                            }
                        },
                        "period": 300,
                        "stat": "Average"
                    },
                    "returnData": true
                }
            ],
            "alarmName": "ServerCpuTooHigh",
            "description": "Goes into alarm when server CPU utilization is too high!",
            "actionsEnabled": true,
            "timestamp": "2022-03-03T17:06:13.757+0000",
            "okActions": [],
            "alarmActions": [],
            "insufficientDataActions": []
        }
    }
}
```

サプレッサーアラーム付き複合アラームの更新

```
{
    "version": "0",
    "id": "4c6f4177-6bd5-c0ca-9f05-b4151c54568b",
    "detail-type": "CloudWatch Alarm Configuration Change",
    "source": "aws.cloudwatch",
    "account": "123456789012",
    "time": "2022-07-14T13:59:56Z",
    "region": "us-east-1",
    "resources": [
        "arn:aws:cloudwatch:us-east-1:123456789012:alarm:ServiceAggregatedAlarm"
    ],
    "detail": {
        "alarmName": "ServiceAggregatedAlarm",
        "operation": "update",
        "state": {
            "actionsSuppressedBy": "WaitPeriod",
```

```
            "value": "ALARM",
            "timestamp": "2022-07-14T13:59:46.425+0000"
        },
        "configuration": {
            "alarmRule": "ALARM(ServerCpuTooHigh) OR
 ALARM(TotalNetworkTrafficTooHigh)",
            "actionsSuppressor": "ServiceMaintenanceAlarm",
            "actionsSuppressorWaitPeriod": 120,
            "actionsSuppressorExtensionPeriod": 360,
            "alarmName": "ServiceAggregatedAlarm",
            "actionsEnabled": true,
            "timestamp": "2022-07-14T13:59:56.290+0000",
            "okActions": [],
            "alarmActions": [],
            "insufficientDataActions": []
        },
        "previousConfiguration": {
            "alarmRule": "ALARM(ServerCpuTooHigh) OR
 ALARM(TotalNetworkTrafficTooHigh)",
            "actionsSuppressor": "ServiceMaintenanceAlarm",
            "actionsSuppressorWaitPeriod": 120,
            "actionsSuppressorExtensionPeriod": 180,
            "alarmName": "ServiceAggregatedAlarm",
            "actionsEnabled": true,
            "timestamp": "2022-07-14T13:59:46.425+0000",
            "okActions": [],
            "alarmActions": [],
            "insufficientDataActions": []
        }
    }
}
```

メトリクスの数式に基づくアラームの削除

```
{

    "version": "0",
    "id": "f171d220-9e1c-c252-5042-2677347a83ed",
    "detail-type": "CloudWatch Alarm Configuration Change",
    "source": "aws.cloudwatch",
    "account": "123456789012",
    "time": "2022-03-03T17:07:13Z",
    "region": "us-east-1",
```

```
    "resources": [
        "arn:aws:cloudwatch:us-east-1:123456789012:alarm:TotalNetworkTrafficTooHigh"
    ],
    "detail": {
        "alarmName": "TotalNetworkTrafficTooHigh",
        "operation": "delete",
        "state": {
            "value": "INSUFFICIENT_DATA",
            "timestamp": "2022-03-03T17:06:17.672+0000"
        },
        "configuration": {
            "evaluationPeriods": 1,
            "threshold": 10000,
            "comparisonOperator": "GreaterThanThreshold",
            "treatMissingData": "ignore",
            "metrics": [{
                    "id": "m1",
                    "metricStat": {
                        "metric": {
                            "namespace": "AWS/EC2",
                            "name": "NetworkIn",
                            "dimensions": {
                                "InstanceId": "i-12345678901234567"
                            }
                        },
                        "period": 300,
                        "stat": "Maximum"
                    },
                    "returnData": false
                },
                {
                    "id": "m2",
                    "metricStat": {
                        "metric": {
                            "namespace": "AWS/EC2",
                            "name": "NetworkOut",
                            "dimensions": {
                                "InstanceId": "i-12345678901234567"
                            }
                        },
                        "period": 300,
                        "stat": "Maximum"
                    },
                    "returnData": false
```

```
                },
                {
                    "id": "e1",
                    "expression": "SUM(METRICS())",
                    "label": "Total Network Traffic",
                    "returnData": true
                }
            ],
            "alarmName": "TotalNetworkTrafficTooHigh",
            "description": "Goes into alarm if total network traffic exceeds 10Kb",
            "actionsEnabled": true,
            "timestamp": "2022-03-03T17:06:17.672+0000",
            "okActions": [],
            "alarmActions": [],
            "insufficientDataActions": []

        }
    }
}
```

サプレッサーアラーム付き複合アラームの削除

```
{
    "version": "0",
    "id": "e34592a1-46c0-b316-f614-1b17a87be9dc",
    "detail-type": "CloudWatch Alarm Configuration Change",
    "source": "aws.cloudwatch",
    "account": "123456789012",
    "time": "2022-07-14T14:00:01Z",
    "region": "us-east-1",
    "resources": [
        "arn:aws:cloudwatch:us-east-1:123456789012:alarm:ServiceAggregatedAlarm"
    ],
    "detail": {
        "alarmName": "ServiceAggregatedAlarm",
        "operation": "delete",
        "state": {
            "actionsSuppressedBy": "WaitPeriod",
            "value": "ALARM",
            "timestamp": "2022-07-14T13:59:46.425+0000"
        },
        "configuration": {
```

```
            "alarmRule": "ALARM(ServerCpuTooHigh) OR
 ALARM(TotalNetworkTrafficTooHigh)",
            "actionsSuppressor": "ServiceMaintenanceAlarm",
            "actionsSuppressorWaitPeriod": 120,
            "actionsSuppressorExtensionPeriod": 360,
            "alarmName": "ServiceAggregatedAlarm",
            "actionsEnabled": true,
            "timestamp": "2022-07-14T13:59:56.290+0000",
            "okActions": [],
            "alarmActions": [],
            "insufficientDataActions": []
        }
    }
}
```

アラームの管理

CloudWatch アラームを編集または削除する

既存のアラームを編集または削除できます。

既存のアラームの名前は変更できません。アラームをコピーし、新しいコピーしたアラームに別の名前を付けることができます。アラームをコピーするには、アラームリストでアラーム名の横にあるチェックボックスをオンにし、[アクション]、[コピー] の順に選択します。

アラームを編集するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで、[Alarms] (アラーム)、[All Alarms] (すべてのアラーム) の順に選択します。
3. アラーム名を指定します。
4. タグを追加、または削除するには、[タグ] タブを選択して、[タグの管理] を選択します。
5. アラームの他の部分を編集するには、[アクション]、[編集] を選択します。

   [Specify metric and conditions (メトリクスと条件の指定)] ページに、選択したメトリクスと統計のグラフや他の情報が表示されます。
6. メトリクスを変更するには、 [編集] を選択し、[すべてのメトリクス] タブを選択して次のいずれかの操作を行います。

- 必要なメトリクスが含まれているサービス名前空間を選択します。表示されるオプションを選択し続けて選択肢を絞り込みます。メトリクスのリストが表示されたら、必要なメトリクスの横にあるチェックボックスをオンにします。
- 検索ボックスに、メトリクス、ディメンション、またはリソース ID の名前を入力し、Enter キーを押します。次に、結果のいずれかを選択する操作を続けて、メトリクスを一覧表示します。必要なメトリクスの横にあるチェックボックスをオンにします。

[メトリクスの選択] を選択します。

7. アラームの他の側面を変更するには、適切なオプションを選択します。アラームが ALARM 状態に移行するためにしきい値を超過する必要があるデータポイントの数を変更したり、欠落データの処理方法を変更したりするには、[Additional configuration (追加設定)] を選択します。
8. [Next] を選択します。
9. [通知] の [Auto Scaling アクション] および [EC2 アクション] で、必要に応じて、アラームがトリガーされたときに実行するアクションを編集します。次いで、[次へ] を選択します。
10. 必要に応じてアラームの説明を変更します。

    既存のアラームの名前は変更できません。アラームをコピーし、新しいコピーしたアラームに別の名前を付けることができます。アラームをコピーするには、アラームリストでアラーム名の横にあるチェックボックスをオンにし、[アクション]、[コピー] の順に選択します。
11. [Next] を選択します。
12. [Preview and create (プレビューして作成)] で、情報と条件が正しいことを確認し、[Update alarm (アラームの更新)] を選択します。

Amazon SNS コンソールを使用して作成された E メール通知リストを更新するには

1. Amazon SNS コンソール (https://console.aws.amazon.com/sns/v3/home) を開きます。
2. ナビゲーションペインで、[トピック] を選択し、通知リスト (トピック) の ARN を選択します。
3. 次のいずれかを行ってください。

    - メールアドレスを追加するには、[サブスクリプションの作成] を選択します。[Protocol (プロトコル)] として [Email (E メール)] を選択します。[エンドポイント] に、新しい受取人のメールアドレスを入力します。[Create subscription] を選択します。
    - メールアドレスを削除するには、[サブスクリプション ID] を選択します。[その他のサブスクリプションの操作]、[サブスクリプションの削除] の順に選択します。
4. [トピックに発行] を選択します。

アラームを削除するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで、[Alarms] を選択します。
3. アラーム名の左側にあるチェックボックスをオンにし、[アクション]、[削除] の順に選択します。
4. [削除] を選択します。

Auto Scaling アラームを非表示にする

AWS Management Console でアラームを表示すると、Amazon EC2 Auto Scaling と Application Auto Scaling の両方に関連するアラームを非表示にすることができます。この機能は、 でのみ使用できますAWS Management Console

Auto Scaling アラームを一時的に非表示にするには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで、[Alarms] (アラーム)、[All alarms] (すべてのアラーム) の順に選択し、[Hide Auto Scaling alarms] (すべてのオートスケーリングアラームを非表示にする) を選択します。

アラームのユースケースと例

以下のセクションでは、一般的な使用例に対するアラームの例とチュートリアルを紹介します。

AWS の予想請求額をモニタリングする請求アラームの作成

Amazon CloudWatch を使用して、 予想 AWS 請求額をモニターリングすることができます。ご使用の AWS アカウントに対する予想請求額のモニターリングを有効にすると、予想請求額が計算されて、メトリクスデータとして CloudWatch に 1 日複数回送信されます。

請求メトリクスデータは米国東部 (バージニア北部) リージョンに保存され、世界全体の請求額として表されます。このデータには、使用した AWS のサービス別の予想請求額と AWS 全体の予想請求額が含まれています。

アカウントへの請求が指定したしきい値を超えると、アラームがトリガーされます。トリガーされるのは、現在の請求がしきい値を超えた場合のみです。当月のそれまでの使用状況に基づく予測は使用されません。

請求がすでにしきい値を超えているときに請求アラームを作成した場合、アラームは直ちに ALARM 状態になります。

> **Note**
>
> 既に請求されている CloudWatch の料金の分析について詳しくは、「CloudWatch の請求とコスト」を参照してください。

タスク

- 請求アラートの有効化
- 請求アラームの作成
- 請求アラームの削除

請求アラートの有効化

予想請求額のアラームを作成する前に、請求アラートを有効にする必要があります。有効にすると、AWS の予想請求額をモニターリングし、請求メトリクスデータを使用してアラームを作成できます。請求アラートを有効にすると、データの収集を無効にできなくなりますが、作成した請求アラームはいつでも削除できます。

初めて予想請求額のモニターリングを有効にした場合、請求データの表示と請求アラートの設定ができるようになるまで約 15 分かかります。

要件

- アカウントのルートユーザー認証情報を使用するか、請求情報を表示するアクセス許可が付与されている IAM ユーザーとしてサインインする必要があります。
- 一括請求 (コンソリデーティッドビリング) のアカウントの場合、支払いアカウントでログインすると、リンクされている各アカウントの請求データを見ることができます。リンクされているそれぞれのアカウントと一括請求アカウントのどちらに対しても、予想請求合計額とサービスごとの予想請求額のデータを見ることができます。
- 一括請求 (コンソリデーティッドビリング) アカウントでは、メンバー連結アカウントのメトリクスは、支払者アカウントが [請求アラートを受け取る] 設定を有効にしている場合にのみキャプ

チャされます。管理/支払者アカウントのアカウントを変更する場合は、新しい管理/支払者アカウントの請求アラートを有効にする必要があります。

- Amazon パートナーネットワーク (APN) アカウントの請求メトリクスは CloudWatch に対して公開されないため、このアカウントを APN の一部にすることはできません。詳細については、「AWS パートナーネットワーク」を参照してください。

予想請求額のモニターリングを有効にするには

1. AWS Billing コンソール https://console.aws.amazon.com/billing/ を開きます。
2. ナビゲーションペインで、[請求設定] を選択します。
3. [アラート設定] で [編集] を選択します。
4. [CloudWatch 請求アラートを受信する] を選択します。
5. [設定を保存] を選択します。

請求アラームの作成

> **Important**
>
> 請求アラームを作成する前に、リージョンを米国東部 (バージニア北部) に設定する必要があります。請求メトリクスデータは、このリージョンに保存され、世界全体の請求額を示します。また、自分のアカウントまたは管理/支払者アカウント (一括請求を使用している場合) で、請求アラートを有効にする必要があります。詳細については、「請求アラートの有効化」を参照してください。

この手順では、AWS の予想請求額が、定義されたしきい値を超えた場合に通知を送信するアラームを作成します。

CloudWatch コンソールを用いて請求アラームを作成するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで、[Alarms] (アラーム) を選択し、[All alarms] (アラームの作成) を選択します。
3. [アラームの作成] を選択します。
4. メトリクスの選択 を選択します。[Browse] (参照) で、[Billing] (請求) を選択してから、[Total Estimated Charge] (概算合計請求額) を選択します。

> **Note**
>
> [請求]/[概算合計請求額] メトリクスが表示されない場合は、請求アラートを有効にして、リージョンを米国東部 (バージニア北部) に変更します。詳細については、「請求アラートの有効化」を参照してください。

5. [EstimatedCharges] メトリクスの横にあるチェックボックスをオンにして、[Select metric] (メトリクスの選択) を選択します。
6. [統計] で、[Maximum] を選択します。
7. [Period] (期間) で、[6 hours] (6 時間) を選択します。
8. [Threshold type] で [静的] を選択します。
9. [Whenever EstimatedCharges is . . .] (EstimatedCharges が次の場合) で、[Greater] (より大きい) を選択します。
10. [次よりも] で、アラームをトリガーさせる値を定義します。例えば、**200** USD などです。

    [EstimatedCharges] のメトリクス値は米ドル (USD) のみで、通貨換算は Amazon Services LLC が行います。詳細については、「AWS Billing とは」を参照してください。

> **Note**
>
> しきい値を定義すると、プレビューグラフに当月の予想請求額が表示されます。

11. [追加設定] を選択し、次の操作を行います。
    - [Datapoints to alarm] (アラームを実行するデータポイント) で、1 個中 1 個 を指定します。
    - [Missing data treatment] (欠落データの処理) で、[Treat missing data as missing] (欠落データを欠落として処理) を選択します。
12. [Next] を選択します。
13. [通知] で、[アラーム状態] が選択されていることを確認します。次に、アラームが ALARM 状態の時に通知される Amazon SNS トピックを選択します。Amazon SNS トピックには E メールアドレスを含めることができるため、請求金額が指定したしきい値を超えたときに E メールを受信できます。

    既存の Amazon SNS トピックを選択するか、新しい Amazon SNS トピックを作成するか、またはトピック ARN を使用して他のアカウントに通知することができます。同じアラーム状態ま

たは異なるアラーム状態について複数の通知を送信する場合は、[Add notification] (通知の追加) を選択します。

14. [Next] を選択します。
15. [Name and description] (名前と説明) で、アラームの名前と説明を入力します。アラーム名には UTF-8 文字のみを使用する必要があり、ASCII 制御文字は使用できません。

    - (オプション) アラームの説明を入力します。説明にはマークダウン形式を含めることができます。マークダウン形式は、CloudWatch コンソールのアラームの [詳細] タブにのみ表示されます。マークダウンは、ランブックや他の内部リソースへのリンクを追加するのに役立ちます。

16. [Next] を選択します。
17. [Preview and create] (プレビューと作成) で、設定が正しいことを確認してから、[Create alarm] (アラームの作成) を選択します。

請求アラームの削除

不要になった請求アラームは削除できます。

請求アラームを削除するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. 必要に応じて、リージョンを米国東部 (バージニア) に変更します。請求メトリクスデータは、このリージョンに保存され、世界全体の請求額を反映します。
3. ナビゲーションペインで、[アラーム]、[すべてのアラーム] の順に選択します。
4. アラームの横にあるチェックボックスをオンにして、[アクション]、[削除] の順に選択します。
5. 確認を求めるメッセージが表示されたら、[Yes、Delete] を選択します。

CPU 使用率アラームの作成

アラームの状態が OK から ALARM に切り替わったら、Amazon SNS を使用して通知を送信する CloudWatch アラームを作成できます。

EC2 インスタンスの平均 CPU 使用率が連続した指定期間のしきい値を超えると、アラームは ALARM 状態に変わります。

を使用した CPU 使用率アラームの設定AWS Management Console

AWS Management Console を使用して CPU 使用率アラームを作成するには、次のステップを実行します。

CPU 使用率に基づいてアラームを作成するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで、[Alarms] (アラーム)、[All Alarms] (すべてのアラーム) の順に選択します。
3. [アラームの作成] を選択します。
4. メトリクスの選択 を選択します。
5. [すべてのメトリクス] タブで、[EC2 metrics (EC2 メトリクス)] を選択します。
6. メトリクスカテゴリー (例: [インスタンス別メトリクス]) を選択します。
7. [InstanceId] 列にリストするインスタンスがあり、メトリクス名] 列に[CPUUtilization] がある行を見つけます。この行の横にあるチェックボックスをオンにして、[メトリクスの選択] を選択します。
8. [メトリクスと条件の指定] の、[統計] で [平均] を選択し、事前定義されたパーセンタイルのいずれかを選択するか、カスタムパーセンタイル (**p95.45** など) を指定します。
9. 期間 (例: **5 minutes**) を選択します。
10. [Conditions (条件)] で、次のように指定します。

    a. [Threshold type (しきい値タイプ)] で [静的] を選択します。

    b. [Whenever CPUUtilization is (CPUUtilization が次の場合)] で、[より大きい] を指定します。[... よりも] で、CPU 使用率がこのパーセンテージを超えた場合に、アラームが ALARM 状態になるようトリガーするしきい値を指定します。たとえば、70 です。

    c. [Additional configuration (追加設定)] を選択します。[Datapoints to alarm (アラームを発生させるデータポイント数)] で、アラームをトリガーするために ALARM 状態を維持する必要がある評価期間 (データポイント) の数を指定します。2 つの値が一致する場合は、該当する数の連続した期間でしきい値を超過したときに ALARM 状態に移行するアラームを作成します。

    N 個中 M 個のアラームを作成するには、2 番目の値よりも小さい数字を最初の値に指定します。詳細については、「アラームの評価」を参照してください。

d. [Missing data treatment (欠落データの処理)] 、一部のデータポイントが欠落しているときのアラームによる対処方法を選択します。詳細については、「CloudWatch アラームの欠落データの処理の設定」を参照してください。

e. モニターリングする統計としてパーセンタイルをアラームで使用している場合は、[サンプル数が少ないパーセンタイル] ボックスが表示されます。これを使用して、サンプル数が少ないケースを評価するか無視するかを選択します。[無視 (アラーム状態を維持する)] を選択すると、サンプル数が少なすぎる場合でも現在のアラーム状態が常に維持されます。詳細については、「パーセンタイルベースの CloudWatch アラームおよび少数のデータサンプル」を参照してください。

11. [Next] を選択します。

12. [通知] で、[アラーム状態] を選択し、アラームが ALARM 状態のときに通知する SNS トピックを選択します。

    同じアラーム状態または複数の異なるアラーム状態について複数の通知を送信するには、[Add notification (通知の追加)] を選択します。

    アラームの通知を送信しない場合は、 [削除] を選択します。

13. 完了したら、[次へ] を選択します。

14. アラームの名前と説明を入力します。次いで、[次へ] を選択します。

    アラーム名には UTF-8 文字のみを使用する必要があり、ASCII 制御文字は使用できません。説明にはマークダウン形式を含めることができます。マークダウン形式は、CloudWatch コンソールのアラームの [詳細] タブにのみ表示されます。マークダウンは、ランブックや他の内部リソースへのリンクを追加するのに役立ちます。

15. [Preview and create (プレビューして作成)] で、情報と条件が正しいことを確認し、[アラームの作成] を選択します。

を使用した CPU 使用率アラームの設定AWS CLI

AWS CLI を使用して CPU 使用率アラームを作成するには、次のステップを実行します。

CPU 使用率に基づいてアラームを作成するには

1. SNS トピックを設定します。詳細については、「Amazon SNS 通知の設定」を参照してください。

2. 以下のように put-metric-alarm コマンドを使用してアラームを作成します。

```
aws cloudwatch put-metric-alarm --alarm-name cpu-mon --alarm-description "Alarm
 when CPU exceeds 70%" --metric-name CPUUtilization --namespace AWS/EC2 --statistic
 Average --period 300 --threshold 70 --comparison-operator GreaterThanThreshold --
dimensions  Name=InstanceId,Value=i-12345678 --evaluation-periods 2 --alarm-actions
 arn:aws:sns:us-east-1:111122223333:my-topic --unit Percent
```

3. set-alarm-state コマンドを使用してアラーム状態の変更を適用することにより、アラームをテストします。

   a. アラームの状態を INSUFFICIENT_DATA から OK に変更します。

   ```
   aws cloudwatch set-alarm-state --alarm-name cpu-mon --state-reason
    "initializing" --state-value OK
   ```

   b. アラームの状態を OK から ALARM に変更します。

   ```
   aws cloudwatch set-alarm-state --alarm-name cpu-mon --state-reason
    "initializing" --state-value ALARM
   ```

   c. アラームに関する通知を受け取ったことを確認します。

E メールを送信するロードバランサーレイテンシーアラームの作成

Classic Load Balancer で 100 ms を超えるレイテンシーをモニターリングする Amazon SNS 通知を設定し、アラームを設定できます。

を使用したレイテンシーアラームの設定AWS Management Console

AWS Management Console を使用してロードバランサーレイテンシーアラームを作成するには、次のステップを実行します。

ロードバランサーのレイテンシーアラームを作成するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで、[Alarms] (アラーム)、[All Alarms] (すべてのアラーム) の順に選択します。
3. [アラームの作成] を選択します。
4. [カテゴリー別の CloudWatch メトリクス] で、[ELB メトリクス] カテゴリーを選択します。
5. [Classic Load Balancer] と [レイテンシー] メトリクスの行を選択します。

6. 統計で、[平均] を選択し、事前定義されたパーセンタイルのいずれかを選択するか、カスタムパーセンタイル (**p95.45** など) を指定します。
7. 期間として [1 分] を選択します。
8. [Next] を選択します。
9. [アラームのしきい値] で、アラームの一意の名前 (**myHighCpuAlarm** など) とアラームの説明 (**Alarm when Latency exceeds 100s** など) を入力します。アラーム名には UTF-8 文字のみを使用する必要があり、ASCII 制御文字は使用できません。

   アラーム名には UTF-8 文字のみを使用する必要があり、ASCII 制御文字は使用できません。説明にはマークダウン形式を含めることができます。マークダウン形式は、CloudWatch コンソールのアラームの [詳細] タブにのみ表示されます。マークダウンは、ランブックや他の内部リソースへのリンクを追加するのに役立ちます。
10. [次の時] の [が:] で [>] を選択し、「**0.1**」と入力します。[を:] に「**3**」と入力します。
11. [追加設定] の下の [欠落データの処理方法] で、[無視 (アラーム状態を維持する)] を選択し、欠落データポイントでアラーム状態の変更がトリガーされないようにします。

    [サンプル数が少ないパーセンタイル] で、[無視 (アラーム状態を維持する)] を選択し、データサンプル数が適切なケースのみがアラームで評価されるようにします。
12. [アクション] の [アラームが次の時:] で、[状態: 警告] を選択します。[通知の送信先] で、既存の SNS トピックを選択するか、新しいトピックを作成します。

    SNS トピックを作成するには、[新しいリスト] を選択します。[通知の送信先] に、SNS トピックの名前 (例: **myHighCpuAlarm**) を入力し、[メールリスト] に、アラームが ALARM 状態に変わったときに通知を受けるメールアドレスをカンマで区切って入力します。各 E メールアドレスに、トピックのサブスクリプションの確認メールが送信されます。通知を送信する前にサブスクリプションを確認する必要があります。
13. [アラームの作成] を選択します。

を使用したレイテンシーアラームの設定AWS CLI

AWS CLI を使用してロードバランサーレイテンシーアラームを作成するには、次のステップを実行します。

ロードバランサーのレイテンシーアラームを作成するには

1. SNS トピックを設定します。詳細については、「Amazon SNS 通知の設定」を参照してください。

2. 以下のように put-metric-alarm コマンドを使用してアラームを作成します。

```
aws cloudwatch put-metric-alarm --alarm-name lb-mon --alarm-description "Alarm
 when Latency exceeds 100s" --metric-name Latency --namespace AWS/ELB --statistic
 Average --period 60 --threshold 100 --comparison-operator GreaterThanThreshold --
dimensions Name=LoadBalancerName,Value=my-server --evaluation-periods 3 --alarm-
actions arn:aws:sns:us-east-1:111122223333:my-topic --unit Seconds
```

3. set-alarm-state コマンドを使用してアラーム状態の変更を適用することにより、アラームをテストします。

   a. アラームの状態を INSUFFICIENT_DATA から OK に変更します。

   ```
   aws cloudwatch set-alarm-state --alarm-name lb-mon --state-reason
    "initializing" --state-value OK
   ```

   b. アラームの状態を OK から ALARM に変更します。

   ```
   aws cloudwatch set-alarm-state --alarm-name lb-mon --state-reason
    "initializing" --state-value ALARM
   ```

   c. アラームに関するメール通知を受け取ったことを確認します。

E メールを送信するストレージスループットアラームの作成

SNS 通知をセットアップし、Amazon EBS が 100 MB スループットを超えたときにトリガーされるアラームを設定できます。

を使用したストレージスループットアラームの設定AWS Management Console

AWS Management Console を使用して、Amazon EBS スループットに基づくアラームを作成するには、次のステップを使用します。

ストレージスループットアラームを作成するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで、[Alarms] (アラーム)、[All Alarms] (すべてのアラーム) の順に選択します。
3. [アラームの作成] を選択します。
4. [EBS メトリクス] で、メトリクスカテゴリーを選択します。

5. ボリュームと VolumeWriteBytes メトリクスがある行を選択します。
6. 統計用には [Average] を選択します。期間用には [5 Minutes] を選択します。[Next] を選択します。
7. [アラームのしきい値] で、アラームの一意の名前 (**myHighWriteAlarm** など) とアラームの説明 (**VolumeWriteBytes exceeds 100,000 KiB/s** など) を入力します。アラーム名には UTF-8 文字のみを使用する必要があり、ASCII 制御文字は使用できません。説明にはマークダウン形式を含めることができます。マークダウン形式は、CloudWatch コンソールのアラームの [詳細] タブにのみ表示されます。マークダウンは、ランブックや他の内部リソースへのリンクを追加するのに役立ちます。
8. [次の時] の [が:] で [>] を選択し、「**100000**」と入力します。[を:] で、連続した期間として「**15**」と入力します。

   しきい値がグラフ化されて [アラームのプレビュー] に表示されます。
9. [追加設定] の下の [欠落データの処理方法] で、[無視 (アラーム状態を維持する)] を選択し、欠落データポイントでアラーム状態の変更がトリガーされないようにします。
10. [アクション] の [アラームが次の時:] で、[状態: 警告] を選択します。[通知の送信先] で、既存の SNS トピックを選択するか、トピックを作成します。

    SNS トピックを作成するには、[新しいリスト] を選択します。[通知の送信先] に、SNS トピックの名前 (例: **myHighCpuAlarm**) を入力し、[メールリスト] に、アラームが ALARM 状態に変わったときに通知を受けるメールアドレスをカンマで区切って入力します。各 E メールアドレスに、トピックのサブスクリプションの確認メールが送信されます。E メールアドレスに通知を送信する前に、サブスクリプションを確認する必要があります。
11. [アラームの作成] を選択します。

を使用したストレージスループットアラームの設定AWS CLI

AWS CLI を使用して、Amazon EBS スループットに基づくアラームを作成するには、次のステップを使用します。

ストレージスループットアラームを作成するには

1. SNS トピックを作成します。詳細については、「Amazon SNS 通知の設定」を参照してください。
2. アラームを作成します。

```
aws cloudwatch put-metric-alarm --alarm-name ebs-mon --alarm-description "Alarm
 when EBS volume exceeds 100MB throughput" --metric-name VolumeReadBytes --
namespace AWS/EBS --statistic Average --period 300 --threshold 100000000 --
comparison-operator GreaterThanThreshold --dimensions Name=VolumeId,Value=my-
volume-id --evaluation-periods 3 --alarm-actions arn:aws:sns:us-
east-1:111122223333:my-alarm-topic --insufficient-data-actions arn:aws:sns:us-
east-1:111122223333:my-insufficient-data-topic
```

3. set-alarm-state コマンドを使用してアラーム状態の変更を適用することにより、アラームをテストします。

   a. アラームの状態を INSUFFICIENT_DATA から OK に変更します。

   ```
   aws cloudwatch set-alarm-state --alarm-name ebs-mon --state-reason
    "initializing" --state-value OK
   ```

   b. アラームの状態を OK から ALARM に変更します。

   ```
   aws cloudwatch set-alarm-state --alarm-name ebs-mon --state-reason
    "initializing" --state-value ALARM
   ```

   c. アラームの状態を ALARM から INSUFFICIENT_DATA に変更します。

   ```
   aws cloudwatch set-alarm-state --alarm-name ebs-mon --state-reason
    "initializing" --state-value INSUFFICIENT_DATA
   ```

   d. アラームに関するメール通知を受け取ったことを確認します。

AWS データベースから Performance Insights カウンターメトリクスのアラームを作成する

CloudWatch には DB_PERF_INSIGHTS Metric Math 関数が含まれています。これを使用して Amazon Relational Database Service および Amazon DocumentDB (MongoDB との互換性あり) から CloudWatch に Performance Insights カウンターメトリクスを取り込むことができます。また、DB_PERF_INSIGHTS は 1 分未満の間隔で DBLoad メトリクスを取り込みます。これらのメトリクスに基づいた CloudWatch アラームを設定することができます。

Amazon RDS Performance Insights については、「Amazon RDS での Performance Insights を使用したDB 負荷のモニタリング」を参照してください。

Amazon DocumentDB Performance Insights の詳細については、「Amazon RDS Performance Insights の使用」を参照してください。

DB_PERF_INSIGHTS 関数に基づくアラームでは、異常検出はサポートされていません。

> **Note**
>
> DB_PERF_INSIGHTS によって 1 分未満の解像度で取得される高解像度メトリクスは、DBLoad メトリック、またはより高い解像度で拡張モニタリングを有効にしている場合はオペレーティングシステムメトリックにのみ適用されます。Amazon RDS の拡張モニタリングの詳細については、「拡張モニタリングを使用した OS メトリクスのモニタリング」を参照してください。
> DB_PERF_INSIGHTS 関数を使用して、高解像度のアラームを作成できます。高解像度アラームの最大評価範囲は 3 時間です。CloudWatch コンソールを使用して、DB_PERF_INSIGHTS 関数で取得したメトリクスを任意の時間範囲でグラフ化できます。

Performance Insights メトリクスに基づいたアラームを作成するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで、[Alarms] (アラーム) を選択し、[All alarms] (アラームの作成) を選択します。
3. [アラームの作成] を選択します。
4. [メトリクスの選択] を選択します。
5. [数式を追加] ドロップダウンを選択し、リストから[データベースパフォーマンスのメトリクス]、[DB_PERF_INSIGHTS] を選択します。

   [DB_PERF_INSIGHTS] を選択すると、数式を適用または編集できる数式ボックスが表示されます。
6. 数式ボックスに DB_PERF_INSIGHTS 数式を入力し、 [適用] を選択します。

   例えば、**`DB_PERF_INSIGHTS('RDS', 'db-ABCDEFGHIJKLMNOPQRSTUVWXY1', 'os.cpuUtilization.user.avg')`**

> **Important**
>
> [DB_PERF_INSIGHTS] の数式を使用するときは、データベースの一意のデータベースリソース ID を指定する必要があります。これはデータベース ID とは異なります。Amazon RDS コンソールでデータベースリソース ID を検索するには、DB インスタンスを選択してその詳細を表示します。そして、[Configuration (設定)] タブを選択します。[設定] セクションに [リソース ID] が表示されます。

DB_PERF_INSIGHTS 関数およびメトリクス計算に使用できるその他の関数の詳細については、「Metric Math 構文と関数」を参照してください。

7. [メトリクスの選択] を選択します。

   [Specify metric and conditions (メトリクスと条件の指定)] ページに、選択した数式に関するグラフや他の情報が表示されます。

8. [**#**が次の時] で、式がしきい値より大きい、より小さい、またはしきい値と等しい必要があるかどうかを指定します。[than... (以下の値)] で、しきい値を指定します。

9. [Additional configuration (追加設定)] を選択します。[Datapoints to alarm (アラームを発生させるデータポイント数)] で、アラームをトリガーするために ALARM 状態を維持する必要がある評価期間 (データポイント) の数を指定します。2 つの値が一致する場合は、該当する数の連続した期間でしきい値を超過したときに ALARM 状態に移行するアラームを作成します。

   N 個中 M 個のアラームを作成するには、2 番目の値よりも小さい数字を最初の値に指定します。詳細については、「アラームの評価」を参照してください。

10. [Missing data treatment (欠落データの処理)] 、一部のデータポイントが欠落しているときのアラームによる対処方法を選択します。詳細については、「CloudWatch アラームの欠落データの処理の設定」を参照してください。

11. [Next (次へ)] を選択します。

12. [通知] で、アラームが ALARM 状態、OK 状態、または INSUFFICIENT_DATA 状態のときに通知するための SNS トピックを選択します。

    同じアラーム状態または複数の異なるアラーム状態について複数の通知を送信するには、[Add notification (通知の追加)] を選択します。

    アラームの通知を送信しない場合は、 [削除] を選択します。

13. アラームに伴って Auto Scaling、EC2、Lambda、または Systems Manager アクションを実行するには、該当するボタンを選択し、アラーム状態と実行するアクションを選択します。Lambda 関数をアラームアクションとして選択する場合、関数名または ARN を指定し、オプションで関数の特定のバージョンを選択できます。

    アラームは、ALARM 状態になったときにのみ、Systems Manager のアクションを実行できます。Systems Manager のアクションの詳細については、「アラームから OpsItems を作成するように CloudWatch を設定する」および「Incident creation」を参照してください。

    > **Note**
    >
    > SSM Incident Manager アクションを実行するアラームを作成するには、特定のアクセス許可が必要です。詳細については、AWS Systems Manager Incident Manager のアイデンティティベースのポリシーの例を参照してください。

14. 完了したら、[次へ] を選択します。
15. アラームの名前と説明を入力します。次いで、[次へ] を選択します。

    アラーム名には UTF-8 文字のみを使用する必要があり、ASCII 制御文字は使用できません。説明にはマークダウン形式を含めることができます。マークダウン形式は、CloudWatch コンソールのアラームの [詳細] タブにのみ表示されます。マークダウンは、ランブックや他の内部リソースへのリンクを追加するのに役立ちます。
16. [Preview and create (プレビューして作成)] で、情報と条件が正しいことを確認し、[アラームの作成] を選択します。

EC2 インスタンスを停止、終了、再起動、または復旧するアラームを作成する

Amazon CloudWatch アラームアクションを使用して、EC2 インスタンスを自動的に停止、終了、再起動、または復旧するアラームを作成できます。停止または終了アクションを使用すると、今後インスタンスを実行する必要がなくなったときにコストを節約できます。再起動アクションを使用すると、これらのインスタンスを自動的に再起動でき、復旧アクションを使用すると、システムで障害が発生した場合に新しいハードウェアで復旧できます。

自動的にインスタンスを停止または終了するシナリオはいくつもあります。例えば、バッチ給与計算処理ジョブまたは科学計算タスクを専用に行うインスタンスを使用している場合が挙げられます。これらのインスタンスは一定期間動作して仕事を完了します。このようなインスタンスは、アイドル状

態 (課金されている状態) にせずに、停止または終了するとコスト削減につながります。アラームアクションの停止と終了の主な違いとして、停止したインスタンスは、後で再実行が必要な場合に簡単に再起動できます。また、同じインスタンス ID とルートボリュームを保持することもできます。しかし、終了したインスタンスを再起動することはできません。代わりに新しいインスタンスを開始する必要があります。

停止、終了、再起動、アクションは、Amazon CloudWatch によって (AWS/EC2 名前空間で) 提供されている基本および詳細モニターリングメトリクスや、「InstanceId=」ディメンションを含んでいるカスタムメトリクスなど、Amazon EC2 インスタンスごとのメトリクスで設定されている任意のアラームに追加できます。ただし、InstanceId 値が実行中の有効な Amazon EC2 インスタンスを参照している場合に限ります。復旧アクションは、`StatusCheckFailed_Instance` を除き、Amazon EC2 のインスタンスごとのメトリクスで設定されているアラームに追加することもできます。

インスタンスを再起動、停止、終了可能な CloudWatch アラームアクションをセットアップするには、サービスにリンクされた IAM ロール AWSServiceRoleForCloudWatchEvents を使用する必要があります。AWSServiceRoleForCloudWatchEvents IAM ロールを使用すると、AWS がお客様に代わってアラームアクションを実行できます。

CloudWatch Events のサービスにリンクされたロールを作成するには、次のコマンドを使用します。

```
aws iam create-service-linked-role --aws-service-name events.amazonaws.com
```

コンソールのサポート

CloudWatch コンソールまたは Amazon EC2 コンソールを使用してアラームを作成できます。このドキュメントの手順では、CloudWatch コンソールを使用します。Amazon EC2 コンソールを使用する手順については、「Linux インスタンス用 Amazon EC2 ユーザーガイド」の「インスタンスを停止、終了、再起動、または復旧するアラームを作成する」を参照してください。

アクセス許可

AWS Identity and Access Management (IAM) アカウントを使用して EC2 アクションまたは Systems Manager OpsItem アクションを実行するアラームを作成または変更する場合は、`iam:CreateServiceLinkedRole` 許可が必要です。

内容

- Amazon CloudWatch アラームへの停止アクションの追加

- Amazon CloudWatch アラームへの終了アクションの追加
- Amazon CloudWatch アラームへの再起動アクションの追加
- Amazon CloudWatch アラームへの復旧アクションの追加
- トリガーされたアラームとアクションの履歴の表示

Amazon CloudWatch アラームへの停止アクションの追加

一定のしきい値に達したときに Amazon EC2 インスタンスを停止するアラームを作成できます。例えば、開発またはテスト用のインスタンスを実行したまま、終了するのを忘れることがたまにあります。平均 CPU 利用率が 24 時間 10% 未満である場合に、インスタンスがアイドル状態で使用されていないという信号を発してトリガーするアラームを作成できます。しきい値、持続時間、期間を二ーズに合わせて調整し、アラームがトリガーされたときにメールを受信するよう SNS 通知を追加できます。

Amazon Elastic Block Store ボリュームをルートデバイスとして使用する Amazon EC2 インスタンスは停止または終了できますが、インスタンスストアをルートデバイスとして使用するインスタンスでは終了のみ行えます。

Amazon CloudWatch コンソールを使用してアイドル状態のインスタンスを停止させるアラームを作成するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで、[アラーム]、[すべてのアラーム] の順に選択します。
3. [アラームの作成] を選択します。
4. [メトリクスの選択] を選択します。
5. AWS 名前空間 の場合は、[EC2] を選択します。
6. 以下の操作を実行します。
    a. [Per-Instance Metrics] (インスタンス別メトリクス) を選択します。
    b. 正しいインスタンスと CPUUtilization メトリクスを含む行のチェックボックスを選択します。
    c. [グラフ化したメトリクス] タブを選択します。
    d. 統計用には [Average] を選択します。
    e. 期間 (例: **1 Hour**) を選択します。
    f. [メトリクスの選択] を選択します。

7. [アラームの定義] ステップでは、以下の作業を行います。

   a. [Conditions (条件)] で、[Static (静的)] を選択します。

   b. [Whenever CPUUtilization is] (CPU 使用率が次の状態であるときは常に) で、[Lower] (より低い) を選択します。

   c. [than] (よりも) で、**10** と入力します。

   d. [Next] を選択します。

   e. [通知] の [通知の送信先] で、既存の SNS トピックを選択するか、新しいトピックを作成します。

      SNS トピックを作成するには、[新しいリスト] を選択します。[通知の送信先] に、SNS トピックの名前を入力します (例: Stop_EC2_Instance)。[メールリスト] に、アラームが ALARM 状態に変わったら通知する E メールアドレスを、カンマ区切りのリストに入力します。各 E メールアドレスに、トピックのサブスクリプションの確認メールが送信されます。E メールアドレスに通知を送信する前に、サブスクリプションを確認する必要があります。

   f. [Add EC2 Action] (EC2 アクションの追加) を選択します。

   g. [Alarm state trigger] (アラーム状態トリガー) で、[In alarm] (アラーム状態) を選択します。[Take the following action] (次のアクションを実行) で、[Stop this instance] (このインスタンスを停止する) を選択します。

   h. [Next] を選択します。

   i. アラームの名前と説明を入力します。名前には ASCII 文字のみを使用します。続いて、[次へ] を選択します。

   j. [Preview and create (プレビューして作成)] で、情報と条件が正しいことを確認し、[アラームの作成] を選択します。

Amazon CloudWatch アラームへの終了アクションの追加

(インスタンスで終了保護が有効になっていない限り)、一定のしきい値に達したときに EC2 インスタンスを自動的に終了させるアラームを作成することができます。たとえば、インスタンスが仕事を終え、再びそのインスタンスを使用する必要がない場合は、インスタンスを削除することをお勧めします。後でインスタンスを使用する可能性がある場合は、インスタンスを削除するのではなく、停止するほうが良いでしょう。インスタンスの終了保護の有効化または無効化については、「Linux インスタンス用 Amazon EC2 ユーザーガイド」の「Enabling Termination Protection for an Instance」(インスタンスの終了保護の有効化) を参照してください。

Amazon CloudWatch コンソールを使用してアイドル状態のインスタンスを終了させるアラームを作成するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで、[Alarms]、[Create Alarm] の順に選択します。
3. [メトリクスの選択] ステップでは、以下の作業を行います。
   a. [EC2 メトリクス] で、[インスタンス別メトリクス] を選択します。
   b. インスタンスおよび CPUUtilization メトリクスがある行を選択します。
   c. 統計用には [Average] を選択します。
   d. 期間 (例: **1 Hour**) を選択します。
   e. [Next] を選択します。
4. [アラームの定義] ステップでは、以下の作業を行います。
   a. [アラームのしきい値] で、アラームの一意の名前 (例: Terminate EC2 instance) と、アラームの説明 (例: Terminate EC2 instance when CPU is idle for too long) を入力します。アラーム名には ASCII 文字のみを使用する必要があります。
   b. [次の時] の [が:] で [<] を選択し、「**10**」と入力します。[期間] で、連続した期間として「**24**」を入力します。

      しきい値がグラフ化されて [アラームのプレビュー] に表示されます。
   c. [通知] の [通知の送信先] で、既存の SNS トピックを選択するか、新しいトピックを作成します。

      SNS トピックを作成するには、[新しいリスト] を選択します。[通知の送信先] に、SNS トピックの名前を入力します (例: Terminate_EC2_Instance)。[メールリスト] に、アラームが ALARM 状態に変わったら通知する E メールアドレスを、カンマ区切りのリストに入力します。各 E メールアドレスに、トピックのサブスクリプションの確認メールが送信されます。E メールアドレスに通知を送信する前に、サブスクリプションを確認する必要があります。
   d. [EC2 アクション] を選択します。
   e. [アラームが次の時] で [状態: 警告] を選択します。[次のアクションを実行] で [このインスタンスの削除] を選択します。
   f. [アラームの作成] を選択します。

Amazon CloudWatch アラームへの再起動アクションの追加

Amazon EC2 インスタンスをモニターリングし、自動的に再起動する Amazon CloudWatch アラームを作成できます。再起動アラームアクションは、インスタンスのヘルスチェックが失敗した場合に推奨されます (システムのヘルスチェックが失敗した場合には、復旧アラームアクションが推奨されます) 。インスタンスの再起動は、オペレーティングシステムの再起動と同等です。ほとんどの場合、インスタンスの再起動には数分しかかかりません。インスタンスを再起動すると、インスタンスは同じホスト上で保持されるため、インスタンスのパブリック DNS 名、プライベート IP アドレス、およびインスタンスストアボリューム上のすべてのデータは保持されます。

インスタンスを再起動しても、インスタンスの停止と再起動とは異なり、新しいインスタンスの課金時間は開始されません。インスタンスの再起動についての詳細は、「Linux インスタンス用 Amazon EC2 ユーザーガイド」の「インスタンスの再起動」を参照してください。

> **Important**
>
> 再起動と復旧アクション間で不具合が発生するのを回避するには、再起動アラームと復旧アラームを同じ検査期間に設定するのを避けます。再起動アラームを各 1 分間の 3 回の評価期間に設定することをお勧めします。

Amazon CloudWatch コンソールを使用してインスタンスを再起動するアラームを作成するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで、[Alarms]、[Create Alarm] の順に選択します。
3. [メトリクスの選択] ステップでは、以下の作業を行います。
   a. [EC2 メトリクス] で、[インスタンス別メトリクス] を選択します。
   b. インスタンスおよび StatusCheckFailed_Instance メトリクスがある行を選択します。
   c. 統計用には [Minimum] を選択します。
   d. 期間 (例: **`1 Minute`**) を選択します。
   e. [Next] を選択します。
4. [アラームの定義] ステップでは、以下の作業を行います。
   a. [アラームのしきい値] で、アラームの一意の名前 (例: Reboot EC2 instance) と、アラームの説明 (例: Reboot EC2 instance when health checks fail) を入力します。アラーム名には ASCII 文字のみを使用する必要があります。

b. [次の時] の [が:] で [>] を選択し、「**0**」と入力します。[期間] で、連続した期間として「**3**」を入力します。

しきい値がグラフ化されて [アラームのプレビュー] に表示されます。

c. [通知] の [通知の送信先] で、既存の SNS トピックを選択するか、新しいトピックを作成します。

SNS トピックを作成するには、[新しいリスト] を選択します。[通知の送信先] に、SNS トピックの名前を入力します (例: Reboot_EC2_Instance)。[メールリスト] に、アラームが ALARM 状態に変わったら通知する E メールアドレスを、カンマ区切りのリストに入力します。各 E メールアドレスに、トピックのサブスクリプションの確認メールが送信されます。E メールアドレスに通知を送信する前に、サブスクリプションを確認する必要があります。

d. [EC2 アクション] を選択します。

e. [アラームが次の時] で [状態: 警告] を選択します。[次のアクションを実行] で [このインスタンスの再起動] を選択します。

f. [アラームの作成] を選択します。

Amazon CloudWatch アラームへの復旧アクションの追加

Amazon EC2 インスタンスをモニターリングする Amazon CloudWatch アラームを作成できます。この機能により、基盤ハードウェアの障害や、AWS による修復を必要とする問題によりインスタンスが正常に機能しなくなった場合に、自動的にそのインスタンスを復旧できます。終了したインスタンスは復旧できません。復旧されたインスタンスは、インスタンス ID、プライベート IP アドレス、Elastic IP アドレス、すべてのインスタンスメタデータを含め、元のインスタンスと同じです。

StatusCheckFailed_System アラームがトリガーされ、復旧アクションが開始されると、アラームを作成したときに選択し、復旧アクションに関連付けた Amazon SNS トピックによって通知されます。インスタンスを復旧する際、インスタンスを再起動するときにインスタンスは移行され、メモリ内にあるデータは失われます。プロセスが完了すると、情報はアラームに設定された SNS トピックに発行されます。この SNS トピックにサブスクライブされるすべてのユーザーは、復旧処理のステータスと、それ以降の手順を含むメールの通知を受け取ります。復旧されたインスタンスでインスタンスが再起動されたことがわかります。

復旧アクションは、StatusCheckFailed_System でのみ使用できます。StatusCheckFailed_Instance では使用できません。

システムステータスチェックの失敗の原因となる問題には、次のようなものがあります。

- ネットワーク接続の喪失
- システム電源の喪失
- 物理ホストのソフトウェアの問題
- ネットワーク到達可能性に影響する、物理ホスト上のハードウェアの問題

復旧アクションは、一部のインスタンスタイプでのみサポートされています。サポートされているインスタンスタイプとその他の要件についての詳細は、「インスタンスの復旧」および「要件」を参照してください。

> **Important**
>
> 再起動と復旧アクション間で不具合が発生するのを回避するには、再起動アラームと復旧アラームを同じ検査期間に設定するのを避けます。復旧アラームを各 1 分間の 2 つの評価期間に設定し、再起動アラームを各 1 分間の 3 つの評価期間に設定することをお勧めします。

Amazon CloudWatch コンソールを使用してインスタンスを復旧するアラームを作成するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで、[Alarms]、[Create Alarm] の順に選択します。
3. [メトリクスの選択] ステップでは、以下の作業を行います。
   a. [EC2 メトリクス] で、[インスタンス別メトリクス] を選択します。
   b. インスタンスおよび StatusCheckFailed_System メトリクスがある行を選択します。
   c. 統計用には [Minimum] を選択します。
   d. 期間 (例: **`1 Minute`**) を選択します。

      > **Important**
      >
      > 再起動と復旧アクション間で不具合が発生するのを回避するには、再起動アラームと復旧アラームを同じ検査期間に設定するのを避けます。復旧アラームを各 1 分間の 2 回の評価期間に設定することをお勧めします。

   e. [Next] を選択します。

4. [アラームの定義] ステップでは、以下の作業を行います。

   a. [アラームのしきい値] で、アラームの一意の名前 (例: Recover EC2 instance) と、アラームの説明 (例: Recover EC2 instance when health checks fail) を入力します。アラーム名には ASCII 文字のみを使用する必要があります。

   b. [次の時] の [が:] で [>] を選択し、「**0**」と入力します。[期間] で、連続した期間として「**2**」を入力します。

   c. [通知] の [通知の送信先] で、既存の SNS トピックを選択するか、新しいトピックを作成します。

      SNS トピックを作成するには、[新しいリスト] を選択します。[通知の送信先] に、SNS トピックの名前を入力します (例: Recover_EC2_Instance)。[メールリスト] に、アラームが ALARM 状態に変わったら通知する E メールアドレスを、カンマ区切りのリストに入力します。各 E メールアドレスに、トピックのサブスクリプションの確認メールが送信されます。E メールアドレスに通知を送信する前に、サブスクリプションを確認する必要があります。

   d. [EC2 アクション] を選択します。

   e. [アラームが次の時] で [状態: 警告] を選択します。[次のアクションを実行] で [このインスタンスの復元] を選択します。

   f. [アラームの作成] を選択します。

トリガーされたアラームとアクションの履歴の表示

Amazon CloudWatch コンソールで、アラームとアクションの履歴を表示できます。Amazon CloudWatch は、過去 30 日分のアラームとアクション履歴を保持します。

トリガーされたアラームとアクションを表示するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで [アラーム] を選択し、アラームを選択します。
3. 直近の状態遷移、および時間とメトリクス値を表示するには、[詳細] を選択します。
4. 直近の履歴のエントリを表示するには、[履歴] を選択します。

アラームとタグ付け

タグとは、リソースの整理や分類に役立つキーと値のペアです。また、特定のタグ値を含むリソースのみにアクセスする、またはそれらのルールを変更する許可をユーザーに付与することで、タグを使用してユーザーアクセス許可を制限することもできます。リソースへのタグ付けに関する一般情報については、「AWS リソースのタグ付け」を参照してください。

次のリストでは、CloudWatch アラームでのタグ付けの仕組みについての詳細を説明しています。

- CloudWatch リソースのタグを設定または更新するには、`cloudwatch:TagResource` アクセス許可を備えたアカウントにサインインする必要があります。例えば、アラームを作成してタグを設定するには、the `cloudwatch:PutMetricAlarm` アクセス許可に加えて `cloudwatch:TagResource` アクセス許可が必要です。CloudWatch リソースを作成または更新する組織内のすべてのユーザーに `cloudwatch:TagResource` アクセス許可があることを確認することをお勧めします。
- タグはタグベースの承認制御に使用できます。例えば、IAM ユーザーまたはロールのアクセス許可に条件を含めることで、タグに基づいて CloudWatch 呼び出しを特定のリソースに制限できます。ただし、次の点に注意してください。
  - 名前が `aws:` で始まるタグは、タグベースの承認制御には使用できません。
  - 複合アラームは、タグベースの承認制御をサポートしていません。

Application Signals

⚠ Application Signals はプレビューリリース中です。この機能についてご意見がありましたら、app-signals-feedback@amazon.com までご連絡ください。

CloudWatch Application Signals を使用して AWS でアプリケーションを自動的に計測することで、現在のアプリケーションの状態をモニタリングし、ビジネス目標に照らして長期的なアプリケーションパフォーマンスを追跡できます。Application Signals は、アプリケーション、サービス、依存関係をアプリケーション中心の統合ビューで表示し、アプリケーションの状態をモニタリングしてトリアージできるようにします。

- Application Signals を有効にすると、アプリケーションからメトリクスとトレースが自動的に収集され、呼び出し量、可用性、レイテンシー、障害、エラーなどの主要なメトリクスが表示されます。カスタムコードを記述したり、ダッシュボードを作成したりしなくても、現在の運用状態や、アプリケーションが長期的なパフォーマンス目標を達成しているかどうかをすばやく確認してトリアージできます。
- Application Signals を使用して、サービスレベル目標 (SLO) を作成し、モニタリングします。Application Signals が収集する新しい標準アプリケーションメトリクスを含む、CloudWatch メトリクスに関連する SLO のステータスを簡単に作成して追跡できます。アプリケーションサービスのサービスレベル指標 (SLI) のステータスをサービスリストとトポロジマップ内で確認して追跡できます。SLO を追跡するアラームを作成したり、Application Signals が収集する新しい標準アプリケーションメトリクスを追跡したりできます。
- Application Signals が自動的に検出するアプリケーショントポロジのマップを参照してください。これには、アプリケーション、依存関係、およびそれらの接続性が視覚的に表示されています。
- Application Signals は、CloudWatch RUM、CloudWatch Synthetics canary、AWS Service Catalog AppRegistry、および Amazon EC2 Auto Scaling と連携し、ダッシュボードやマップ内にクライアントページ、Synthetics canary、アプリケーション名を表示します。

毎日のアプリケーションモニタリングに Application Signals を使用する

毎日のアプリケーションモニタリングの一環として、CloudWatch コンソール内で Application Signals を使用します。

1. サービスのサービスレベル目標 (SLO) を作成した場合は、サービスレベル目標 (SLO) ページから始めてください。これにより、最も重要なサービスとオペレーションの状態をすぐに把握できます。SLO のサービス名またはオペレーション名を選択すると、サービスの詳細ページが開き、問題のトラブルシューティング時に詳細なサービス情報が表示されます。
2. サービスページを開くと、すべてのサービスの概要が表示され、障害率またはレイテンシーが最も高いサービスをすばやく確認できます。SLO を作成した場合は、サービスのテーブルを見て、どのサービスに異常なサービスレベル指標 (SLI) があるかを確認してください。特定のサービスが異常状態にある場合は、サービスを選択してサービスの詳細ページを開き、サービスオペレーション、依存関係、Synthetics Canary、およびクライアントリクエストを確認します。グラフ内のポイントを選択して、相関関係のあるトレースを表示すると、運用上の問題の根本原因をトラブルシューティングして特定できます。
3. 新しいサービスがデプロイされたり、依存関係が変更されたりした場合は、Service Map を開いてアプリケーショントポロジを調べてください。クライアント、Synthetics Canary、サービス、依存関係の関係を示すアプリケーションのマップを参照してください。SLI の状態をすばやく確認したり、呼び出し量、障害率、レイテンシーなどの主要なメトリクスを表示したり、サービスの詳細ページでドリルダウンして詳細情報を確認したりできます。

Application Signals を使用すると、料金が発生します。CloudWatch の料金の詳細については、Amazon CloudWatch の料金をご覧ください。

> **Note**
>
> CloudWatch Synthetics、CloudWatch RUM、または CloudWatch Evident を使用するために、Application Signals を有効にする必要はありません。ただし、Synthetics と CloudWatch RUM を Application Signals と連携させると、これらの機能を一緒に使用した場合にメリットが得られます。

サポートされている言語とアーキテクチャ

現在、Application Signals は Java および Python アプリケーションをサポートしています。

Application Signals は、Amazon EKS、Amazon ECS、および Amazon EC2 でサポートされ、テストされています。Amazon EKS クラスターでは、サービスとクラスターの名前が自動的に検出されます。他のアーキテクチャでは、Application Signals に対してそれらのサービスを有効にするときに、サービスと環境の名前を指定する必要があります。

Amazon EC2 で Application Signals を有効にする手順は、CloudWatch エージェントと AWS Distro for OpenTelemetry をサポートするすべてのアーキテクチャで機能する必要があります。ただし、この手順は Amazon ECS と Amazon EC2 以外のアーキテクチャではテストされていません。

サポートされるリージョン

今回のプレビューリリースでは、Application Signals は以下のリージョンでサポートされています。

- 米国東部 (バージニア北部)
- 米国東部 (オハイオ)
- 米国西部 (オレゴン)
- アジアパシフィック (シドニー)
- アジアパシフィック (東京)
- 欧州 (アイルランド)

Preview SDK

SDK のプレビュー版はダウンロード可能です。

> **Warning**
>
> API オペレーションとパラメータは、Application Signals が一般公開される前に変更されることがあります。これらの変更は重大な変更になる可能性があります。SDK のプレビュー版を本稼働目的で使用しないでください。

Preview SDK をインストールするには、まず AWS CLI バージョン 2 の最新バージョンをインストールまたは更新してください。詳細については、「AWS CLI の最新バージョンを使用してインストールまたは更新を行う」を参照してください。

次に、次のコマンドを使用して Amazon S3 バケットから SDK zip ファイルをダウンロードし、その内容を抽出します。各 SDK zip ファイルには、SDK での手順と API ドキュメントが含まれています。

> **Note**
>
> SDK は複数のプログラミング言語で提供されており、これらのプログラミング言語のいずれかで Application Signals API を使用できます。ただし、アプリケーションの自動計測による

Application Signals へのデータ送信は、Java および Python アプリケーションでのみサポートされます。

- Java V2 SDK: `aws s3 cp s3://application-signals-preview-sdk/awsJavaSdkV2.zip ./`
- JavaScript V3 SDK: `aws s3 cp s3://application-signals-preview-sdk/jsSdkV3.zip ./`
- JavaScript V2 SDK: `aws s3 cp s3://application-signals-preview-sdk/jsSdkV2.zip ./`
- Python SDK: `aws s3 cp s3://application-signals-preview-sdk/pythonSdk.zip ./`
- Kotlin SDK: `aws s3 cp s3://application-signals-preview-sdk/kotlin.zip ./`
- Android SDK: `aws s3 cp s3://application-signals-preview-sdk/andriod.zip ./`
- C++ SDK: `aws s3 cp s3://application-signals-preview-sdk/awsCppSdk.zip ./`
- PHP SDK: `aws s3 cp s3://application-signals-preview-sdk/awsSdkPhp.zip ./`
- Ruby SDK: `aws s3 cp s3://application-signals-preview-sdk/awsSdkRuby.zip ./`
- Go V2 SDK: `aws s3 cp s3://application-signals-preview-sdk/awsSdkGoV2.zip ./`
- Go V1 SDK: `aws s3 cp s3://application-signals-preview-sdk/go.zip ./`
- iOS SDK: `aws s3 cp s3://application-signals-preview-sdk/iOS.zip ./`

トピック

Application Signals に必要な権限

⚠ Application Signals は Amazon CloudWatch のプレビューリリースであり、変更される可能性があります。

このセクションでは、Application Signals を有効化、管理、運用するために必要な権限について説明します。

Application Signals を有効化および管理するための権限

Application Signals を管理するには、次のアクセス許可でサインオンする必要があります。

```
{
 "Version": "2012-10-17",
 "Statement": [
    {
        "Sid": "CloudWatchApplicationSignalsFullAccessPermissions",
        "Effect": "Allow",
        "Action": "application-signals:*",
        "Resource": "*"
    },
    {
        "Sid": "CloudWatchApplicationSignalsAlarmsPermissions",
        "Effect": "Allow",
        "Action": [
            "cloudwatch:DescribeAlarms"
        ],
        "Resource": "*"
    },
    {
        "Sid": "CloudWatchApplicationSignalsMetricsPermissions",
        "Effect": "Allow",
        "Action": [
            "cloudwatch:GetMetricData",
            "cloudwatch:ListMetrics"
        ],
        "Resource": "*"
    },
    {
        "Sid": "CloudWatchApplicationSignalsLogGroupPermissions",
```

```
        "Effect": "Allow",
        "Action": [
            "logs:StartQuery",
            "logs:DescribeMetricFilters"
        ],
        "Resource": "arn:aws:logs:*:*:log-group:/aws/application-signals/data:*"
    },
    {
        "Sid": "CloudWatchApplicationSignalsLogsPermissions",
        "Effect": "Allow",
        "Action": [
            "logs:GetQueryResults",
            "logs:StopQuery"
        ],
        "Resource": "*"
    },
    {
        "Sid": "CloudWatchApplicationSignalsSyntheticsPermissions",
        "Effect": "Allow",
        "Action": [
            "synthetics:DescribeCanaries",
            "synthetics:DescribeCanariesLastRun",
            "synthetics:GetCanaryRuns"
        ],
        "Resource": "*"
    },
    {
        "Sid": "CloudWatchApplicationSignalsRumPermissions",
        "Effect": "Allow",
        "Action": [
            "rum:BatchCreateRumMetricDefinitions",
            "rum:BatchDeleteRumMetricDefinitions",
            "rum:BatchGetRumMetricDefinitions",
            "rum:GetAppMonitor",
            "rum:GetAppMonitorData",
            "rum:ListAppMonitors",
            "rum:PutRumMetricsDestination",
            "rum:UpdateRumMetricDefinition"
        ],
        "Resource": "*"
    },
    {
        "Sid": "CloudWatchApplicationSignalsXrayPermissions",
        "Effect": "Allow",
```

```
        "Action": [
            "xray:GetTraceSummaries"
        ],
        "Resource": "*"
    },
    {
        "Sid": "CloudWatchApplicationSignalsPutMetricAlarmPermissions",
        "Effect": "Allow",
        "Action": "cloudwatch:PutMetricAlarm",
        "Resource": [
            "arn:aws:cloudwatch:*:*:alarm:SLO-AttainmentGoalAlarm-*",
            "arn:aws:cloudwatch:*:*:alarm:SLO-WarningAlarm-*",
            "arn:aws:cloudwatch:*:*:alarm:SLI-HealthAlarm-*"
        ]
    },
    {
        "Sid": "CloudWatchApplicationSignalsCreateServiceLinkedRolePermissions",
        "Effect": "Allow",
        "Action": "iam:CreateServiceLinkedRole",
        "Resource": "arn:aws:iam::*:role/aws-service-role/application-
signals.cloudwatch.amazonaws.com/AWSServiceRoleForCloudWatchApplicationSignals",
        "Condition": {
            "StringLike": {
                "iam:AWSServiceName": "application-signals.cloudwatch.amazonaws.com"
            }
        }
    },
    {
        "Sid": "CloudWatchApplicationSignalsGetRolePermissions",
        "Effect": "Allow",
        "Action": "iam:GetRole",
        "Resource": "arn:aws:iam::*:role/aws-service-role/application-
signals.cloudwatch.amazonaws.com/AWSServiceRoleForCloudWatchApplicationSignals"
    },
    {
        "Sid": "CloudWatchApplicationSignalsSnsWritePermissions",
        "Effect": "Allow",
        "Action": [
            "sns:CreateTopic",
            "sns:Subscribe"
        ],
        "Resource": "arn:aws:sns:*:*:cloudwatch-application-signals-*"
    },
    {
```

```
        "Sid": "CloudWatchApplicationSignalsSnsReadPermissions",
        "Effect": "Allow",
        "Action": "sns:ListTopics",
        "Resource": "*"
    }
 ]
}
```

Amazon EC2、Kubernetes、カスタムアーキテクチャのいずれかで Application Signals を有効にするには、「カスタムセットアップによって、他のプラットフォームでも Application Signals を有効にする」を参照してください。Amazon CloudWatch Observability EKS アドオンを使用して Amazon EKS で Application Signals を有効化し管理するには、次のアクセス許可が必要です。

> **Important**
>
> これらの権限には、Resource "*" のある iam:PassRole と Resource "*" のある eks:CreateAddon が含まれます。これらは強力な権限なので、付与する際には注意が必要です。

```
{
"Version": "2012-10-17",
    "Statement": [
        {
"Sid": "CloudWatchApplicationSignalsEksAddonManagementPermissions",
            "Effect": "Allow",
            "Action": [
                "eks:AccessKubernetesApi",
                "eks:CreateAddon",
                "eks:DescribeAddon",
                "eks:DescribeAddonConfiguration",
                "eks:DescribeAddonVersions",
                "eks:DescribeCluster",
                "eks:DescribeUpdate",
                "eks:ListAddons",
                "eks:ListClusters",
                "eks:ListUpdates",
                "iam:ListRoles",
                "iam:PassRole"
            ],
            "Resource": "*"
```

```
        },
        {
"Sid":
 "CloudWatchApplicationSignalsEksCloudWatchObservabilityAddonManagementPermissions",
            "Effect": "Allow",
            "Action": [
                "eks:DeleteAddon",
                "eks:UpdateAddon"
            ],
            "Resource": "arn:aws:eks:*:*:addon/*/amazon-cloudwatch-observability/*"
        }
    ]
}
```

Application Signals ダッシュボードには、お使いの SLO が関連付けられている AWS Service Catalog AppRegistry アプリケーションが表示されます。SLO のページでこれらのアプリケーションを表示するには、次のアクセス許可が必要になります。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "CloudWatchApplicationSignalsTaggingReadPermissions",
            "Effect": "Allow",
            "Action": "tag:GetResources",
            "Resource": "*"
        }
    ]
}
```

Application Signals の運用

Application Signals を使用してサービスや SLO を監視するサービスオペレーターは、以下の読み取り専用のアクセス許可を持つアカウントにサインオンする必要があります。

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

```
                "application-signals:BatchGet*",
                "application-signals:Get*",
                "application-signals:List*"
            ],
            "Resource": "*"
        },
        {
            "Sid": "CloudWatchApplicationSignalsGetRolePermissions",
            "Effect": "Allow",
            "Action": "iam:GetRole",
            "Resource": "arn:aws:iam::*:role/aws-service-role/application-
signals.cloudwatch.amazonaws.com/AWSServiceRoleForCloudWatchApplicationSignals"
        },
        {
        "Sid": "CloudWatchApplicationSignalsLogGroupPermissions",
        "Effect": "Allow",
        "Action": [
            "logs:StartQuery",
            "logs:DescribeMetricFilters"
        ],
        "Resource": "arn:aws:logs:*:*:log-group:/aws/application-signals/data:*"
    },
    {
        "Sid": "CloudWatchApplicationSignalsLogsPermissions",
        "Effect": "Allow",
        "Action": [
            "logs:GetQueryResults",
            "logs:StopQuery"
        ],
        "Resource": "*"
    },
        {
            "Sid": "CloudWatchApplicationSignalsAlarmsReadPermissions",
            "Effect": "Allow",
            "Action": [
                "cloudwatch:DescribeAlarms"
            ],
            "Resource": "*"
        },
        {
            "Sid": "CloudWatchApplicationSignalsMetricsReadPermissions",
            "Effect": "Allow",
            "Action": [
                "cloudwatch:GetMetricData",
```

```
                "cloudwatch:ListMetrics"
            ],
            "Resource": "*"
        },
        {
            "Sid": "CloudWatchApplicationSignalsSyntheticsReadPermissions",
            "Effect": "Allow",
            "Action": [
                "synthetics:DescribeCanaries",
                "synthetics:DescribeCanariesLastRun",
                "synthetics:GetCanaryRuns"
            ],
            "Resource": "*"
        },
        {
            "Sid": "CloudWatchApplicationSignalsRumReadPermissions",
            "Effect": "Allow",
            "Action": [
                "rum:BatchGetRumMetricDefinitions",
                "rum:GetAppMonitor",
                "rum:GetAppMonitorData",
                "rum:ListAppMonitors"
            ],
            "Resource": "*"
        },
        {
            "Sid": "CloudWatchApplicationSignalsXrayReadPermissions",
            "Effect": "Allow",
            "Action": [
                "xray:GetTraceSummaries"
            ],
            "Resource": "*"
        }
    ]
}
```

Application Signals のダッシュボードで、SLO が関連付けられている AWS Service Catalog AppRegistry アプリケーションを確認するには、次のアクセス許可が必要になります。

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

```
            "Sid": "CloudWatchApplicationSignalsTaggingReadPermissions",
            "Effect": "Allow",
            "Action": "tag:GetResources",
            "Resource": "*"
        }
    ]
}
```

Amazon CloudWatch Observability EKS アドオンを使用して Amazon EKS で Application Signals をチェックするには、次のアクセス許可が必要になります。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "CloudWatchApplicationSignalsEksReadPermissions",
            "Effect": "Allow",
            "Action": [
                "eks:ListAddons",
                "eks:ListClusters"
            ],
            "Resource": "*"
        },
        {
            "Sid": "CloudWatchApplicationSignalsEksDescribeAddonReadPermissions",
            "Effect": "Allow",
            "Action": [
                "eks:DescribeAddon"
            ],
            "Resource": "arn:aws:eks:*:*:addon/*/amazon-cloudwatch-observability/*"
        }
    ]
}
```

Application Signals を有効にする

⚠ Application Signals はプレビューリリース中です。この機能についてご意見がありましたら、app-signals-feedback@amazon.com までご連絡ください。

このセクションのトピックでは、使用環境で CloudWatch Application Signals を有効にする方法について説明します。Application Signals は、コンソールを使用したセットアップワークフローで運用される Amazon EKS クラスターに対応しており、Amazon EC2 といった他のプラットフォームのカスタムセットアッププロセスでもサポートされています。

トピック

Application Signals のサポート対象システム

⚠ Application Signals はプレビューリリース中です。この機能についてご意見がありましたら、app-signals-feedback@amazon.com までご連絡ください。

Application Signals は、Amazon EKS、Amazon ECS、および Amazon EC2 でサポートされ、テストされています。Amazon EC2 で Application Signals を有効にする手順は、CloudWatch エージェントと AWS Distro for OpenTelemetry をサポートするすべてのプラットフォームに適用されますが、その他のプラットフォームではテストされていません。

Java の互換性

Application Signals は Java アプリケーションをサポートしていますが、AWS Distro for OpenTelemetry と同じ Java ライブラリとフレームワークもサポートしています。詳細については、「Supported libraries, frameworks, application servers, and JVMs」を参照してください。

JVM バージョン 8、11、17 がサポートされています。

Python の互換性

Application Signals は、AWS Distro for OpenTelemetry と同じライブラリとフレームワークをサポートしています。詳細については、opentelemetry-python-contrib の「サポート対象パッケージ」を参照してください。

Python バージョン 3.8 以降がサポートされています。

Python アプリケーションの Application Signals を有効にする前に、以下の考慮事項に注意してください。

- コンテナ化されたアプリケーションの一部は、PYTHONPATH 環境変数がないことが原因でアプリケーションが起動しなくなることがあります。これを解決するには、PYTHONPATH 環境変数をアプリケーションの作業ディレクトリの場所に設定します。これは OpenTelemetry の自動計測に関する既知の問題によるものです。この問題の詳細については、「Python autoinstrumentation setting of PYTHONPATH is not compliant」を参照してください。
- Django アプリケーションには、OpenTelemetry Python ドキュメントで概説されている追加の必須設定があります。
  - `--noreload` フラグを使用すると、自動リロードを防ぐことができます。
  - Django アプリケーションの `settings.py` ファイルの場所に `DJANGO_SETTINGS_MODULE` 環境変数を設定します。これにより、OpenTelemetry がユーザーの Django 設定に正しくアクセスして統合できるようになります。

OpenTelemetry の互換性に関する考慮事項

⚠ Application Signals はプレビューリリース中です。この機能についてご意見がありましたら、app-signals-feedback@amazon.com までご連絡ください。

アプリケーションで CloudWatch Application Signals を利用できるようにするには、既存のパフォーマンスモニタリングソリューションをアプリケーションから完全に削除しておくことをお勧めします。これには、計測コードと設定の削除も含まれます。

Application Signals では、OpenTelemetry の計測機能を使用しますが、既にある OpenTelemetry の計測機能や設定との互換性は保証されません。最良のシナリオであれば、カスタムメトリクスなどの OpenTelemetry 機能の一部を維持できる場合がありますが、詳細については、次のセクションを参照してください。

OpenTelemetry を使用している場合の考慮事項

残りのセクションでは、アプリケーションで OpenTelemetry を使用している方を対象に、Application Signals との互換性を確保するための重要な情報を紹介します。

- アプリケーションで Application Signals を有効にする前に、OpenTelemetry に基づいた自動計算エージェントの挿入をアプリケーションから削除する必要があります。これにより、設定の競合を回避できます。互換性のある OpenTelemetry API と Application Signals を使用して、手動計測の使用を継続することができます。
- 手動計測によってアプリケーションからカスタムスパンまたはメトリクスを生成している場合、計測の複雑さによっては、Application Signals を有効にした後に、データ生成の停止など、望ましくない動作が見られる可能性があります。OpenTelemetry で利用可能な設定の一部 (本セクションで後述する表内の設定を除く) を使用すると、既存のメトリクスまたはスパンの望ましい動作を維持できる場合があります。こうした設定の詳細については、OpenTelemetry ドキュメントの「SDK Configuration」を参照してください。

  例えば、OTEL_EXPORTER_OTLP_METRICS_ENDPOINT 設定と自己管理型の OpenTelemetry Collector インスタンスを使用することで、カスタムメトリクスを望ましい宛先に引き続き送信できる場合があります。
- 環境変数やシステムプロパティの中には、Application Signals で使用してはならないものもあれば、次の表のガイダンスに従っている限り使用できるものもあります。詳細については、表内を参照してください。

環境変数	Application Signals での推奨設定
一般的な環境変数	
OTEL_SDK_DISABLED	true に設定しないでください。
OTEL_TRACES_EXPORTER	otlp に設定する必要があります。
OTEL_EXPORTER_OTLP_ENDPOINT	使用しないでください。
OTEL_EXPORTER_OTLP_TRACES_ENDPOINT	使用しないでください。
OTEL_ATTRIBUTE_COUNT_LIMIT	設定する場合は、CloudWatch Application Signals によってスパン属性をさらに 10 個ほど追加できるよう、十分に大きな値を指定する必要があります。

環境変数	Application Signals での推奨設定
OTEL_PROPAGATORS	設定する場合は、最終トレースを考慮して xray を指定する必要があります。
OTEL_TRACES_SAMPLER	設定する場合は、xray を指定して、一元化された X-Ray サンプリングを使用する必要があります。 ローカルサンプリングを使用するには、これを parentbased_traceidratio に設定し、OTEL_TRACES_SAMPLER_ARG でサンプリングレートを指定します。
OTEL_TRACES_SAMPLER_ARG	一元化された X-Ray トレースサンプルをデフォルト設定で使用している場合は、この変数は使用しないでください。 上記ではなくローカルサンプリングを使用している場合は、この変数にサンプリングレートを設定します。例えば、サンプリングレートが 5% の場合、0.05 と指定します。
Java 固有の環境変数	
OTEL_JAVA_ENABLED_RESOURCE_PROVIDERS	設定する場合は、AWS のリソース検出機能も指定する必要があります。
Python 固有の環境変数	
OTEL_PYTHON_CONFIGURATOR	使用する場合は、aws_configurator に設定する必要があります。
OTEL_PYTHON_DISTRO	使用する場合は、aws_distro に設定する必要があります。

Amazon EKS クラスターで Application Signals を有効にする

⚠ Application Signals はプレビューリリース中です。この機能についてご意見がありましたら、app-signals-feedback@amazon.com までご連絡ください。

CloudWatch Application Signals は、Amazon EKS クラスターで稼働している Java および Python アプリケーションに対応しています。Amazon EKS クラスターのアプリケーション向けに Application Signals を有効にする場合、次の 2 つのオプションを使用できます。

- 既存の Amazon EKS クラスターで稼働するアプリケーション向けに Application Signals を有効にするには、「独自のサービスを使用して Amazon EKS クラスターで Application Signals を有効にする」のステップを実行します。
- サンプルアプリケーションを使用して非本番環境で Application Signals を試すには、「サンプルアプリケーションを使用して新規 Amazon EKS クラスターで Application Signals を有効にする」の手順を実行します。このワークフローでは、AWS が提供するスクリプトを使用して Amazon EKS クラスターを新規作成し、Application Signals が有効になっているサンプルアプリケーションをインストールします。これにより、Application Signals が備えるエンドツーエンドの機能を確認し、テストできます。

トピック

独自のサービスを使用して Amazon EKS クラスターで Application Signals を有効にする

⚠ Application Signals はプレビューリリース中です。この機能についてご意見がありましたら、app-signals-feedback@amazon.com までご連絡ください。

既存の Amazon EKS クラスターで稼働するアプリケーションで CloudWatch Application Signals を有効にするには、このセクションの手順を実行します。

> **Important**
>
> Application Signals を有効にするアプリケーションで OpenTelemetry を使用している場合は、Application Signals を有効にする前に「OpenTelemetry の互換性に関する考慮事項」を参照してください。

既存の Amazon EKS クラスターで稼働するアプリケーション向けに Application Signals を有効にするには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで [Services (サービス)] を選択します。
3. このアカウントで Application Signals をまだ有効にしていない場合は、サービスの検出に必要なアクセス権限を Application Signals に付与する必要があります。このためには、次の操作を行います。この操作はアカウントごとに 1 回のみ必要です。

   a. [サービスの検出を開始] を選択します。

   b. チェックボックスを選択し、[サービスの検出を開始] を選択します。

      このステップをアカウントで初めて完了すると、[AWSServiceRoleForCloudWatchApplicationSignals] サービスリンクロールが作成されます。このロールによって、Application Signals に次のアクセス権限が付与されます。

      - `xray:GetServiceGraph`
      - `logs:StartQuery`
      - `logs:GetQueryResults`
      - `cloudwatch:GetMetricData`
      - `cloudwatch:ListMetrics`
      - `tag:GetResources`

      このロールの詳細については、「CloudWatch Application Signals のサービスリンクロールのアクセス許可」を参照してください。
4. [Application Signals を有効にする] を選択します。
5. [プラットフォームの指定] では、[EKS] を選択します。
6. [EKS クラスターを選択] では、Application Signals を有効にするクラスターを選択します。

7. このクラスターで Amazon CloudWatch Observability EKS アドオンが有効になっていない場合は、有効にするよう求められます。このような場合は、次の操作を行います。

   a. [CloudWatch Observability EKS アドオンの追加] を選択します。Amazon EKS コンソールが表示されます。

   b. [Amazon CloudWatch Observability] のチェックボックスをオンにして [次へ] を選択します。

      CloudWatch Observability EKS アドオンにより、Application Signals と CloudWatch Container Insights の両方で、Amazon EKS のオブザーバビリティが向上します。コンテナインサイトの詳細については、Container Insightsを参照してください。

   c. インストールするアドオンの最新バージョンを選択します。

   d. アドオンに使用する IAM ロールを選択します。[ノードから継承] を選択した場合は、ワーカーノードで使用する IAM ロールに適切なアクセス権限をアタッチします。*my-worker-node-role* を Kubernetes ワーカーノードが使用する IAM ロールに置き換えます。

      ```
      aws iam attach-role-policy \
      --role-name my-worker-node-role \
      --policy-arn arn:aws:iam::aws:policy/CloudWatchAgentServerPolicy \
      --policy-arn arn:aws:iam::aws:policy/AWSXRayWriteOnlyAccess
      ```

   e. アドオンを使用するサービスロールを作成する場合は、「Amazon CloudWatch Observability EKS アドオンを使用して CloudWatch エージェントをインストールする」を参照してください。

   f. [次へ] を選択し、画面に表示される情報を確認して [作成] を選択します。

   g. 次の画面で、[CloudWatch Application Signals を有効にする] を選択して CloudWatch コンソールに戻り、この有効化のプロセスを終了します。

8. アプリケーションを Application Signals で有効にするには、2 つのオプションがあります。一貫性を保つために、クラスターごとに 1 つのオプションを選択することをお勧めします。

   - [コンソール] オプションの方が簡単です。この方法を使用すると、ポッドはただちに再起動します。
   - [マニフェストファイルに注釈を付ける] メソッドを使用すると、ポッドがいつ再起動するかをより細かく制御でき、モニタリングを一元化したくない場合は、より分散的な方法でモニタリングを管理できます。

Console

[コンソール] オプションでは、Amazon CloudWatch Observability EKS アドオンの詳細設定を使用して、サービスの Application Signals を設定します。アドオンの詳細については、「(オプション) その他の設定」を参照してください。

ワークロードと名前空間のリストが表示されない場合は、このクラスターでそれらを表示するための適切なアクセス許可があることを確認してください。詳細については、「必要なアクセス許可」を参照してください。

単一のワークロードをモニタリングすることも、名前空間全体をモニタリングすることもできます。

単一のワークロードをモニタリングするには:

1. モニタリングするワークロードの横にあるチェックボックスをオンにします。
2. ワークロードの言語を選択します。Python アプリケーションの場合は、次に進む前にアプリケーションが必要な前提条件を満たしていることを確認してください。詳細については、「Application Signals を有効にしたが、その後、Python アプリケーションを起動できない」を参照してください。
3. [完了] をクリックします。Amazon CloudWatch オブザーバビリティ EKS アドオンは、AWS Distro for OpenTelemetry 自動計測 (ADOT) SDK をすぐにポッドに挿入し、ポッドの再起動をトリガーしてアプリケーションメトリクスとトレースの収集を可能にします。

名前空間全体をモニタリングするには:

1. モニタリングする名前空間の横にあるチェックボックスをオンにします。
2. ワークロードの言語を選択します。これは、現在デプロイされているか、将来デプロイされるかに関係なく、この名前空間のすべてのワークロードに適用されます。Python アプリケーションの場合は、次に進む前にアプリケーションが必要な前提条件を満たしていることを確認してください。詳細については、「Application Signals を有効にしたが、その後、Python アプリケーションを起動できない」を参照してください。
3. [完了] をクリックします。Amazon CloudWatch オブザーバビリティ EKS アドオンは、AWS Distro for OpenTelemetry 自動計測 (ADOT) SDK をすぐにポッドに挿入し、

ポッドの再起動をトリガーしてアプリケーションメトリクスとトレースの収集を可能にします。

別の Amazon EKS クラスターで Application Signals を有効にするには、[サービス] 画面から [Application Signals を有効にする] を選択します。

Annotate manifest file

CloudWatch コンソールの [サービスのモニタリング] セクションには、クラスターのマニフェスト YAML にアノテーションの追加が必要との説明があります。このアノテーションを追加すると、アプリケーションで自動的に計測が開始され、メトリクス、トレース、ログが Application Signals に送信されます。

アノテーションには次の 2 つのオプションがあります。

- ワークロードへのアノテーション: クラスター内の 1 つのワークロードに対し自動的に計測を開始します。
- 名前空間へのアノテーション: 選択した名前空間にデプロイしたすべてのワークロードに対し自動的に計測を開始します。

これらのオプションのどちらかを選択し、次の適切な手順に従います。

- 単一のワークロードにアノテーションを付けるには:
  1. [ワークロードへのアノテーション] を選択します。
  2. 次のいずれかの行をワークロードマニフェストの `PodTemplate` セクションに貼り付けます。
     - Java ワークロードの場合:`annotations: instrumentation.opentelemetry.io/inject-java: "true"`
     - Python ワークロードの場合:`annotations: instrumentation.opentelemetry.io/inject-python: "true"`

       Python アプリケーションには、追加で必要な設定があります。詳細については、「Application Signals を有効にしたが、その後、Python アプリケーションを起動できない」を参照してください。
  3. ターミナルで、`kubectl apply -f` *`your_deployment_yaml`* と入力して変更を適用します。
- 名前空間内のすべてのワークロードにアノテーションを付けるには:

1. [名前空間へのアノテーション] を選択します。
2. 次のいずれかの行を名前空間マニフェストファイルのメタデータセクションに貼り付けます。名前空間に Java と Python の両方のワークロードが含まれている場合は、これらの行を両方とも名前空間マニフェストファイルに貼り付けます。
   - 名前空間に Java ワークロードがある場合:`annotations: instrumentation.opentelemetry.io/inject-java: "true"`
   - 名前空間に Python ワークロードがある場合:`annotations: instrumentation.opentelemetry.io/inject-python: "true"`

     Python アプリケーションには、追加で必要な設定があります。詳細については、「Application Signals を有効にしたが、その後、Python アプリケーションを起動できない」を参照してください。
3. ターミナルで、`kubectl apply -f` *`your_namespace_yaml`* と入力して変更を適用します。
4. ターミナルで、名前空間のすべてのポッドを再起動するコマンドを入力します。デプロイメントのワークロードを再起動するコマンド例を次にしめします: `kubectl rollout restart deployment -n` *`namespace_name`*

9. [完了後にサービスを表示] を選択します。Application Signals のビューが開き、Application Signals によって収集されているデータが表示されます。データが表示されるまでに数分かかる場合があります。

   別の Amazon EKS クラスターで Application Signals を有効にするには、[サービス] 画面から [Application Signals を有効にする] を選択します。

   [サービス] ビューの詳細については、「Application Signals を使用したアプリケーションの運用状態のモニタリング」を参照してください。

> **Note**
>
> Application Signals の Python アプリケーションを有効にする際に留意すべきいくつかの考慮事項を確認しました。詳細については、「Application Signals を有効にしたが、その後、Python アプリケーションを起動できない」を参照してください。

サンプルアプリケーションを使用して新規 Amazon EKS クラスターで Application Signals を有効にする

⚠ Application Signals はプレビューリリース中です。この機能についてご意見がありましたら、app-signals-feedback@amazon.com までご連絡ください。

サンプルアプリケーションで CloudWatch Application Signals を試した後に、独自のアプリケーションを計測するには、このセクションの手順に従います。次の手順では、スクリプトを使用して Amazon EKS クラスターを作成して、サンプルアプリケーションをインストールします。その後、サンプルアプリケーションを計測し Application Signals と連携させます。

ここでは、Spring を使用する「Pet Clinic」アプリケーションを例に取ります。これは 4 つのマイクロサービスで構成され、これらのサービスは Amazon EC2 上の Amazon EKS で稼働しています。このサービスでは、Application Signals を有効にするスクリプトによってクラスターで Java または Python 自動計測エージェントを利用できるようにします。

要件

- 現在、Application Signals でモニタリング可能なのは Java および Python アプリケーションのみです。
- 対象のインスタンスには AWS CLI がインストールされている必要があります。AWS CLI バージョン 2 をお勧めしますが、バージョン 1 でも問題ありません。AWS CLI のインストールについては、「AWS CLI の最新バージョンを使用してインストールまたは更新を行う」で詳しく確認できます。
- このセクションのスクリプトは Linux 環境と macOS 環境での実行を目的としています。Windows インスタンスの場合は、AWS Cloud9 環境を使用してこれらのスクリプトを実行すると良いでしょう。AWS Cloud9 の詳細については、「AWS Cloud9 とは」をご参照ください。
- サポート対象バージョンの `kubectl` をインストールします。Amazon EKS クラスターコントロールプレーンのマイナーバージョンの差が 1 以内のバージョンの `kubectl` を使用する必要があります。例えば、1.26 の `kubectl` クライアントは、Kubernetes 1.25、1.26、1.27 のクラスターで動作します。既存の Amazon EKS クラスターには、`kubectl` の AWS 認証情報の設定が必要となる場合があります。詳細については、「Amazon EKS クラスターの `kubeconfig` ファイルを作成または更新する」を参照してください。

- `eksctl` をインストールします。`eksctl` と AWS の通信では AWS CLI が使用されるため、eksctl にも AWS CLI と同じ AWS 認証情報が使用されます。詳細については、「Amazon EMR on EKS のセットアップ」で `eksctl` のインストールまたは更新を確認してください。
- `jq` をインストールします。`jq` は、Application Signals を有効にするスクリプトの実行に必要となります。詳細については、「Download jq」を参照してください。

ステップ 1: スクリプトをダウンロードする

サンプルアプリケーションを使用して CloudWatch Application Signals を設定するためのスクリプトをダウンロードするには、zip 形式の GitHub プロジェクトファイルをローカルドライブにダウンロードして解凍するか、GitHub プロジェクトのクローンを作成します。

プロジェクトのクローンを作成するには、ターミナルウィンドウを開き、作業ディレクトリで次の Git コマンドを入力します。

```
git clone https://github.com/aws-observability/application-signals-demo.git
```

ステップ 2: サンプルアプリケーションをビルドしてデプロイする

サンプルアプリケーションイメージをビルドしてプッシュするには、こちらの手順に従います。

ステップ 3: Application Signals とサンプルアプリケーションをデプロイし、有効にする

次の手順を完了する前に、「サンプルアプリケーションを使用して新規 Amazon EKS クラスターで Application Signals を有効にする」に記載の要件を満たしていることを確認してください。

Application Signals とサンプルアプリケーションをデプロイし、有効にするには

1. オンボーディングスクリプトを解凍したローカルターミナルで、次のコマンドを入力します。*`new-cluster-name`* は、新規クラスターの名前に置き換えます。*`region-name`* は、`us-west-1` などの AWS リージョン名に置き換えます。

   このコマンドによって、新規 Amazon EKS クラスターで稼働するサンプルアプリケーションがセットアップされ、Application Signals も有効になります。

   ```
   # assuming the current working directory is 'onboarding'
   # this script sets up a new cluster, enables Application Signals, and deploys the
   # sample application
   cd application-signals-demo/scripts/eks/appsignals/one-step && ./setup.sh new-
   cluster-name region-name
   ```

セットアップスクリプトの実行には約 30 分かかります。この間に、次の処理が行われます。

- 指定したリージョンに Amazon EKS クラスターを新規作成する。
- Application Signals に必要な IAM アクセス権限 (`arn:aws:iam::aws:policy/AWSXrayWriteOnlyAccess` と `arn:aws:iam::aws:policy/CloudWatchAgentServerPolicy`) を作成する。
- Application Signals を有効にします。このために、CloudWatch エージェントをインストールし、CloudWatch メトリクスと X-Ray トレースを考慮してサンプルアプリケーションの自動計測を設定します。
- PetClinic Spring サンプルアプリケーションを同じ Amazon EKS クラスターにデプロイする。
- `pc-add-vist`、`pc-create-owners`、`pc-visit-pet`、`pc-visit-vet`、`pc-clinic-traffic` という 5 つの CloudWatch Synthetics canary を作成します。これらの canary を 1 分間隔で実行して、サンプルアプリケーションの合成トラフィックを生成し、Synthetics canary の結果が Application Signals に表示されるようにします。
- 次の 4 つの名前を使用して、PetClinic アプリケーションのサービスレベル目標 (SLO) を作成します。
  - 所有者検索の可用性
  - 所有者検索のレイテンシー
  - 所有者登録の可用性
  - 所有者登録のレイテンシー
- 次のアクセス権限を Application Signals に付与するカスタム信頼ポリシーを使用して、必要な IAM ロールを作成します。
  - `cloudwatch:PutMetricData`
  - `cloudwatch:GetMetricData`
  - `xray:GetServiceGraph`
  - `logs:StartQuery`
  - `logs:GetQueryResults`

2. (オプション) PetClinic サンプルアプリケーションのソースコードを参照したい場合は、ルートフォルダを確認してください。

```
- application-signals-demo
  - spring-petclinic-admin-server
  - spring-petclinic-api-gateway
  - spring-petclinic-config-server
```

```
  - spring-petclinic-customers-service
  - spring-petclinic-discovery-server
  - spring-petclinic-vets-service
  - spring-petclinic-visits-service
```

3. デプロイした PetClinic サンプルアプリケーションを表示するには、次のコマンドを実行して URL を検索します。

```
kubectl get ingress
```

ステップ 4: サンプルアプリケーションをモニタリングする

前のセクションのステップを実行し、Amazon EKS クラスターの作成とサンプルアプリケーションのデプロイが完了したら、Application Signals を使用してアプリケーションをモニタリングできます。

> **Note**
>
> Application Signals コンソールで入力を行えるようになるには、サンプルアプリケーションへのトラフィックが発生していなければなりません。そのため、前のステップの途中で、そうしたトラフィックを生成する CloudWatch Synthetics canary を作成しました。

サービスの正常性をモニタリングする

これを有効にすると、CloudWatch Application Signals によってサービスのリストが自動的に検出され入力されます。追加の設定は必要ありません。

検出されたサービスのリストを表示し、それらの正常性をモニタリングするには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで、[Application Signals]、[サービス] の順に選択します。
3. サービス、そのオペレーション、依存関係を表示するには、リストにある対象のサービス名を選択します。

   アプリケーションを最重視したこの統合ビューでは、ユーザーとサービスとの間でどのようなやり取りが生じているかを全体的に把握できます。これによって、パフォーマンスに異常が見られた場合に問題の優先順位付けを行えます。[サービス] ビューの詳細については「Application Signals を使用したアプリケーションの運用状態のモニタリング」を参照してください。

4. [サービスオペレーション] タブを選択すると、そのサービスのオペレーションに関する標準アプリケーションメトリクスが表示されます。オペレーションの例には、サービスが呼び出す API オペレーションなどがあります。

   そのサービスオペレーションの 1 つについてグラフを表示するには、対象のオペレーション名を選択します。

5. [依存関係] タブを選択すると、アプリケーションの依存関係に加え、各依存関係の重要なアプリケーションメトリクスが表示されます。依存関係の例には、アプリケーションが呼び出す AWS サービスやサードパーティサービスなどがあります。

6. サービスの詳細ページから相関トレースを表示するには、表の上にある 3 つのグラフのいずれかでデータポイントを選択します。これにより、その期間でフィルタリングされたトレースが新規ペインに表示されます。これらのトレースは、選択したグラフに基づいてソートおよびフィルタリングされています。例えば、[レイテンシー] グラフを選択した場合、トレースはサービスの応答時間でソートされます。

7. CloudWatch コンソールのナビゲーションペインで [SLO] を選択します。スクリプトを使用してサンプルアプリケーション用に作成した SLO が表示されます。SLO の詳細については、「サービスレベル目標 (SLO)」を参照してください。

ステップ 5: (オプション) クリーンアップする

Application Signals のテストが終了したら、Amazon が提供するスクリプトを使用して、アカウントでサンプルアプリケーション用に作成したアーティファクトをクリーンアップし、削除できます。クリーンアップするには、次のコマンドを入力します。*new-cluster-name* をサンプルアプリケーション用に作成したクラスターの名前に、また、*region-name* を AWS リージョンの名前 (us-west-1 など) に置き換えます。

```
cd application-signals-demo/scripts/eks/appsignals/one-step && ./cleanup.sh new-
cluster-name region-name
```

カスタムセットアップによって、他のプラットフォームでも Application Signals を有効にする

⚠ Application Signals はプレビューリリース中です。この機能についてご意見がありましたら、app-signals-feedback@amazon.com までご連絡ください。

CloudWatch Application Signals を Amazon EKS 以外のプラットフォームで有効にするには、このセクションのカスタムセットアップのステップを実行します。これらのアーキテクチャでは、CloudWatch エージェントと AWS Distro for OpenTelemetry をご自身でインストールし設定します。

これらのアーキテクチャの Application Signals では、サービス、そのクラスター、ホストの名前が自動検出されないため、これらの名前をカスタムセットアップの際に指定する必要があります。ここで指定した名前が Application Signals ダッシュボードに表示されることになります。

トピック

カスタムセットアップを使用して Amazon ECS で Application Signals を有効にする

⚠ Application Signals はプレビューリリース中です。この機能についてご意見がありましたら、app-signals-feedback@amazon.com までご連絡ください。

Amazon ECS で稼働するアプリケーションを CloudWatch Application Signals にオンボーディングするには、次のカスタムセットアップの手順を実行します。CloudWatch エージェントと AWS Distro for OpenTelemetry は、ご自身でインストールし、設定します。

Amazon ECS クラスターの Application Signals では、サービス名やサービスが稼働しているクラスターの名前が自動検出されません。これらの名前をカスタムセットアップの際に指定する必要があります。ここで指定した名前が Application Signals ダッシュボードに表示されることになります。

⚠ Important

awsvpc ネットワークモードのみサポートされています。

ステップ 1: アカウントで Application Signals を有効にする

このアカウントで Application Signals をまだ有効にしていない場合は、サービスの検出に必要なアクセス権限を Application Signals に付与する必要があります。このためには、次の操作を行います。この操作はアカウントごとに 1 回のみ必要です。

特定のアプリケーション向けに Application Signals を有効にするには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで [Services (サービス)] を選択します。
3. [サービスの検出を開始] を選択します。
4. チェックボックスを選択し、[サービスの検出を開始] を選択します。

   このステップをアカウントで初めて完了すると、[AWSServiceRoleForCloudWatchApplicationSignals] サービスリンクロールが作成されます。このロールによって、Application Signals に次のアクセス権限が付与されます。

   - `xray:GetServiceGraph`
   - `logs:StartQuery`
   - `logs:GetQueryResults`
   - `cloudwatch:GetMetricData`
   - `cloudwatch:ListMetrics`
   - `tag:GetResources`

   このロールの詳細については、「CloudWatch Application Signals のサービスリンクロールのアクセス許可」を参照してください。

ステップ 2: IAM ロールを作成する

IAM ロールを 2 つ作成する必要があります。これらのロールが作成済みであっても、アクセス権限の追加が必要となる場合があります。

- ECS タスクロール – コンテナは、このロールを使用して実行します。アクセス権限は、アプリケーションに必要な権限の他に、CloudWatchAgentServerPolicy と AWSXRayWriteOnlyAccess を指定する必要があります。
- ECS タスク実行ロール – Amazon ECS は、このロールを使用してコンテナを起動および実行します。このロールを既に作成している場合は、[AmazonSSMReadOnlyAccess]、

[AmazonECSTaskExecutionRolePolicy]、および [CloudWatchAgentServerPolicy] ポリシーをアタッチします。

使用する Amazon ECS のより機密性の高いデータを保存する必要がある場合は、「機密データの指定」を参照してください。

IAM ロールの作成についての詳細は、「IAM ロールの作成」を参照してください。

ステップ 3: CloudWatch エージェント設定を準備する

最初に、Application Signals を有効にしてエージェント設定を準備します。これを行うには、`/tmp/ecs-cwagent.json` という名前のローカルファイルを作成します。

```
{
  "traces": {
    "traces_collected": {
      "app_signals": {}
    }
  },
  "logs": {
    "metrics_collected": {
      "app_signals": {}
    }
  }
}
```

次に、この設定を SSM Parameter Store にアップロードします。これを行うには、次のコマンドを入力します。ファイル内の *$REGION* を実際のリージョン名に置き換えます。

```
aws ssm put-parameter \
--name "ecs-cwagent" \
--type "String" \
--value "`cat /tmp/ecs-cwagent.json`" \
--region "$REGION"
```

ステップ 4: CloudWatch エージェントを使用してアプリケーションを計測する

次のステップは、CloudWatch Application Signals のアプリケーションを計測することです。

Java

CloudWatch エージェントを使用して Amazon ECS で稼働するアプリケーションを計測するには

1. 最初に、バインドマウントを指定します。次のステップでは、このボリュームを使用してコンテナ間でファイルを共有します。このバインドマウントは、そのステップの後半で使用します。

```
"volumes": [
  {
    "name": "opentelemetry-auto-instrumentation"
  }
]
```

2. CloudWatch エージェントのサイドカー定義を追加します。これを行うには、`ecs-cwagent` という新規コンテナをアプリケーションのタスク定義に追加します。*`$REGION`* は実際のリージョン名に置き換えてください。また、Amazon Elastic Container Registry にある最新の CloudWatch コンテナイメージへのパスに置き換えます。詳細については、Amazon ECR の「cloudwatch-agent」を参照してください。

```
{
  "name": "ecs-cwagent",
  "image": "$IMAGE",
  "essential": true,
  "secrets": [
    {
      "name": "CW_CONFIG_CONTENT",
      "valueFrom": "ecs-cwagent"
    }
  ],
  "logConfiguration": {
    "logDriver": "awslogs",
    "options": {
      "awslogs-create-group": "true",
      "awslogs-group": "/ecs/ecs-cwagent",
      "awslogs-region": "$REGION",
      "awslogs-stream-prefix": "ecs"
    }
  }
}
```

3. アプリケーションのタスク定義に新規コンテナ `init` を追加します。*`$IMAGE`* は、AWS Distro for OpenTelemetry Amazon ECR イメージリポジトリにある最新イメージに置き換えてください。

```
{
  "name": "init",
  "image": "$IMAGE",
  "essential": false,
  "command": [
    "cp",
    "/javaagent.jar",
    "/otel-auto-instrumentation/javaagent.jar"
  ],
  "mountPoints": [
    {
      "sourceVolume": "opentelemetry-auto-instrumentation",
      "containerPath": "/otel-auto-instrumentation",
      "readOnly": false
    }
  ]
}
```

4. 次の環境変数をアプリケーションコンテナに追加します。詳細については、以下を参照してください。

環境変数	Application Signals を有効にする設定
`OTEL_RESOURCE_ATTRIBUTES`	`$SVC_NAME` をアプリケーションの名前に置き換えます。これは、Application Signals ダッシュボードにアプリケーション名として表示されます。 `$HOST_ENV` をアプリケーションが稼働しているホスト環境に置き換えてください。これは、アプリケーションの [ホスト元] 環境として Application Signals ダッシュボードに表示されます。

環境変数	Application Signals を有効にする設定
OTEL_AWS_APP_SIGNALS_ENABLED	Application Signals の SpanMetricsProcessor を有効にするには、true に設定します。
OTEL_METRICS_EXPORTER	他のメトリクスエクスポーターを無効にするには none に設定します。
OTEL_AWS_APP_SIGNALS_EXPORTER_ENDPOINT	CloudWatch サイドカーにメトリクスを送信するには、http://127.0.0.1:4315 に設定します。
OTEL_EXPORTER_OTLP_TRACES_ENDPOINT	トレースを CloudWatch サイドカーに送信するには、http://127.0.0.1:4315 に設定します。
OTEL_TRACES_SAMPLER	X-Ray をトレースサンプラーとして定義します。
OTEL_PROPAGATORS	プロパゲーターの 1 つとして X-Ray を追加します。
JAVA_TOOL_OPTIONS	AWS Distro for OpenTelemetry Java エージェントを組み込みます。

5. この手順のステップ 1 で定義したボリューム opentelemetry-auto-instrumentation をマウントします。

   Java アプリケーションの場合は、以下を使用してください。

```
{
  "name": "app",
   ...
  "environment": [
    {
      "name": "OTEL_RESOURCE_ATTRIBUTES",
      "value": "aws.hostedin.environment=$HOST_ENV,service.name=$SVC_NAME"
    },
    {
      "name": "OTEL_AWS_APP_SIGNALS_ENABLED",
```

```
      "value": "true"
    },
    {
      "name": "OTEL_METRICS_EXPORTER",
      "value": "none"
    },
    {
      "name": "JAVA_TOOL_OPTIONS",
      "value": " -javaagent:/otel-auto-instrumentation/javaagent.jar"
    },
    {
      "name": "OTEL_AWS_APP_SIGNALS_EXPORTER_ENDPOINT",
      "value": "http://127.0.0.1:4315"
    },
    {
      "name": "OTEL_TRACES_SAMPLER",
      "value": "xray"
    },
    {
      "name": "OTEL_EXPORTER_OTLP_TRACES_ENDPOINT",
      "value": "http://127.0.0.1:4315"
    },
    {
      "name": "OTEL_PROPAGATORS",
      "value": "tracecontext,baggage,b3,xray"
    }
  ],
  "mountPoints": [
    {
      "sourceVolume": "opentelemetry-auto-instrumentation",
      "containerPath": "/otel-auto-instrumentation",
      "readOnly": false
    }
  ]
}
```

Python

Python アプリケーションの Application Signals を有効にする前に、以下の考慮事項に注意してください。

- コンテナ化されたアプリケーションの一部は、PYTHONPATH 環境変数がないことが原因でアプリケーションが起動しなくなることがあります。これを解決するには、PYTHONPATH 環境変数をアプリケーションの作業ディレクトリの場所に設定します。これは OpenTelemetry の自動計測に関する既知の問題によるものです。この問題の詳細については、「Python autoinstrumentation setting of PYTHONPATH is not compliant」を参照してください。
- Django アプリケーションには、OpenTelemetry Python ドキュメントで概説されている追加の必須設定があります。
  - `--noreload` フラグを使用すると、自動リロードを防ぐことができます。
  - Django アプリケーションの `settings.py` ファイルの場所に `DJANGO_SETTINGS_MODULE` 環境変数を設定します。これにより、OpenTelemetry がユーザーの Django 設定に正しくアクセスして統合できるようになります。

CloudWatch エージェントを使用して Amazon ECS で稼働する Python アプリケーションを計測するには

1. 最初に、バインドマウントを指定します。次のステップでは、このボリュームを使用してコンテナ間でファイルを共有します。このバインドマウントは、そのステップの後半で使用します。

```
"volumes": [
  {
    "name": "opentelemetry-auto-instrumentation-python"
  }
]
```

2. CloudWatch エージェントのサイドカー定義を追加します。これを行うには、`ecs-cwagent` という新規コンテナをアプリケーションのタスク定義に追加します。*`$REGION`* は実際のリージョン名に置き換えてください。また、Amazon Elastic Container Registry にある最新の CloudWatch コンテナイメージへのパスに置き換えます。詳細については、Amazon ECR の「cloudwatch-agent」を参照してください。

```
{
  "name": "ecs-cwagent",
  "image": "$IMAGE",
  "essential": true,
  "secrets": [
    {
      "name": "CW_CONFIG_CONTENT",
```

```
      "valueFrom": "ecs-cwagent"
    }
  ],
  "logConfiguration": {
    "logDriver": "awslogs",
    "options": {
      "awslogs-create-group": "true",
      "awslogs-group": "/ecs/ecs-cwagent",
      "awslogs-region": "$REGION",
      "awslogs-stream-prefix": "ecs"
    }
  }
}
```

3. アプリケーションのタスク定義に新規コンテナ `init` を追加します。*$IMAGE* は、AWS Distro for OpenTelemetry Amazon ECR イメージリポジトリにある最新イメージに置き換えてください。

```
{
    "name": "init",
    "image": "$IMAGE",
    "essential": false,
    "command": [
        "cp",
        "-a",
        "/autoinstrumentation/.",
        "/otel-auto-instrumentation-python"
    ],
    "mountPoints": [
        {
            "sourceVolume": "opentelemetry-auto-instrumentation-python",
            "containerPath": "/otel-auto-instrumentation-python",
            "readOnly": false
        }
    ]
}
```

4. 次の環境変数をアプリケーションコンテナに追加します。詳細については、以下を参照してください。

環境変数	Application Signals を有効にする設定
OTEL_RESOURCE_ATTRIBUTES	$SVC_NAME をアプリケーションの名前に置き換えます。これは、Application Signals ダッシュボードにアプリケーション名として表示されます。 $HOST_ENV をアプリケーションが稼働しているホスト環境に置き換えてください。これは、アプリケーションの [ホスト元] 環境として Application Signals ダッシュボードに表示されます。
OTEL_AWS_APP_SIGNALS_ENABLED	Application Signals の SpanMetricsProcessor を有効にするには、true に設定します。
OTEL_METRICS_EXPORTER	他のメトリクスエクスポーターを無効にするには none に設定します。
OTEL_EXPORTER_OTLP_PROTOCOL	HTTP を使用して CloudWatch にメトリクスとトレースを送信するには、http/protobuf に設定します。
OTEL_AWS_APP_SIGNALS_EXPORTER_ENDPOINT	CloudWatch サイドカーにメトリクスを送信するには、http://127.0.0.1:4316/v1/metrics に設定します。
OTEL_EXPORTER_OTLP_TRACES_ENDPOINT	トレースを CloudWatch サイドカーに送信するには、http://127.0.0.1:4316/v1/traces に設定します。
OTEL_TRACES_SAMPLER	X-Ray をトレースサンプラーとして定義します。
OTEL_PROPAGATORS	プロパゲーターの 1 つとして X-Ray を追加します。

環境変数	Application Signals を有効にする設定
OTEL_PYTHON_DISTRO	ADOT Python 計測を使用するには、aws_distro に設定します。
OTEL_PYTHON_CONFIGURATOR	ADOT Python コンフィギュレーションを使用するには、aws_configuration に設定します。
PYTHONPATH	$APP_PATH をコンテナ内のアプリケーションの作業ディレクトリの場所に置き換えます。これは Python インタープリターがアプリケーションモジュールを見つけるために必要です。
DJANGO_SETTINGS_MODULE	Django アプリケーションでのみ必要です。これは Django アプリケーションの settings.py ファイルの場所に設定してください。$PATH_TO_SETTINGS を置換します。

5. この手順のステップ 1 で定義したボリューム opentelemetry-auto-instrumentation-python をマウントします。

   Python アプリケーションの場合は、以下を使用します。

```
{
    "name": "app",
    ...
    "environment": [
        {
            "name": "PYTHONPATH",
            "value": "/otel-auto-instrumentation-python/opentelemetry/
instrumentation/auto_instrumentation:$APP_PATH:/otel-auto-instrumentation-
python"
        },
        {
            "name": "OTEL_EXPORTER_OTLP_PROTOCOL",
            "value": "http/protobuf"
        },
        {
```

```
            "name": "OTEL_TRACES_SAMPLER",
            "value": "xray"
        },
        {
            "name": "OTEL_TRACES_SAMPLER_ARG",
            "value": "endpoint=http://localhost:2000"
        },
        {
            "name": "OTEL_LOGS_EXPORTER",
            "value": "none"
        },
        {
            "name": "OTEL_PYTHON_DISTRO",
            "value": "aws_distro"
        },
        {
            "name": "OTEL_PYTHON_CONFIGURATOR",
            "value": "aws_configurator"
        },
        {
            "name": "OTEL_EXPORTER_OTLP_TRACES_ENDPOINT",
            "value": "http://localhost:4316/v1/traces"
        },
        {
            "name": "OTEL_AWS_APP_SIGNALS_EXPORTER_ENDPOINT",
            "value": "http://localhost:4316/v1/metrics"
        },
        {
            "name": "OTEL_METRICS_EXPORTER",
            "value": "none"
        },
        {
            "name": "OTEL_AWS_APP_SIGNALS_ENABLED",
            "value": "true"
        },
        {
            "name": "OTEL_RESOURCE_ATTRIBUTES",
            "value": "aws.hostedin.environment=$HOST_ENV,service.name=$SVC_NAM"
        },
        {
            "name": "DJANGO_SETTINGS_MODULE",
            "value": "$PATH_TO_SETTINGS.settings"
        }
    ],
```

```
    "mountPoints": [
        {
            "sourceVolume": "opentelemetry-auto-instrumentation-python",
            "containerPath": "/otel-auto-instrumentation-python",
            "readOnly": false
        }
    ]
}
```

ステップ 5: アプリケーションをデプロイする

タスク定義のリビジョンを新規作成し、アプリケーションクラスターにデプロイします。新規作成したタスクに次の 3 つのコンテナが表示されます。

- `init`
- `ecs-cwagent`
- `app`

カスタムセットアップを使用して Amazon EC2 などのプラットフォームで Application Signals を有効にする

⚠ Application Signals はプレビューリリース中です。この機能についてご意見がありましたら、app-signals-feedback@amazon.com までご連絡ください。

Amazon EKS 以外のアーキテクチャ (Amazon EC2 など) で稼働するアプリケーションの場合は、CloudWatch エージェントと AWS Distro for OpenTelemetry をご自身でインストールし設定します。カスタム Application Signals セットアップで有効になったこれらのアーキテクチャでは、サービス名や、サービスが稼働しているホスト名またはクラスター名が Application Signals によって自動検出されません。これらの名前をカスタムセットアップの際に指定する必要があります。ここで指定した名前が Application Signals ダッシュボードに表示されることになります。

次の手順は Amazon EC2 インスタンスでのテストが完了していますが、AWS Distro for OpenTelemetry をサポートする他のアーキテクチャにも適用可能と思われます。

要件

- Application Signals のサポートを受けるには、CloudWatch エージェントと AWS Distro for OpenTelemetry エージェントの両方で最新バージョンを使用する必要があります。
- 対象のインスタンスには AWS CLI がインストールされている必要があります。AWS CLI バージョン 2 をお勧めしますが、バージョン 1 でも問題ありません。AWS CLI のインストールについては、「AWS CLI の最新バージョンを使用してインストールまたは更新を行う」で詳しく確認できます。

> **Important**
>
> Application Signals を有効にするアプリケーションで OpenTelemetry を使用している場合は、Application Signals を有効にする前に「OpenTelemetry の互換性に関する考慮事項」を参照してください。

ステップ 1: アカウントで Application Signals を有効にする

このアカウントで Application Signals をまだ有効にしていない場合は、サービスの検出に必要なアクセス権限を Application Signals に付与する必要があります。このためには、次の操作を行います。この操作はアカウントごとに 1 回のみ必要です。

特定のアプリケーション向けに Application Signals を有効にするには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで [Services (サービス)] を選択します。
3. [サービスの検出を開始] を選択します。
4. チェックボックスを選択し、[サービスの検出を開始] を選択します。

   このステップをアカウントで初めて完了すると、[AWSServiceRoleForCloudWatchApplicationSignals] サービスリンクロールが作成されます。このロールによって、Application Signals に次のアクセス権限が付与されます。

   - `xray:GetServiceGraph`
   - `logs:StartQuery`
   - `logs:GetQueryResults`
   - `cloudwatch:GetMetricData`
   - `cloudwatch:ListMetrics`

- `tag:GetResources`

このロールの詳細については、「CloudWatch Application Signals のサービスリンクロールのアクセス許可」を参照してください。

ステップ 2: CloudWatch エージェントをダウンロードして起動する

Amazon EC2 インスタンスで Application Signals を有効にする手順内で、CloudWatch エージェントをインストールするには

1. 最新バージョンの CloudWatch エージェントをインスタンスにダウンロードします。インスタンスに CloudWatch エージェントを既にインストールしていても、更新が必要となる場合があります。2023 年 11 月 30 日以降にリリースされたエージェントのバージョンのみ、CloudWatch Application Signals に対応しています。

   CloudWatch エージェントのダウンロードについては、「CloudWatch エージェントパッケージをダウンロードする」を参照してください。

2. CloudWatch エージェントを起動する前に、その設定を行い、Application Signals を有効化できるようにします。次の例は、EC2 ホストのメトリクスとトレースの両方に対し Application Signals を有効にする CloudWatch エージェント設定を示しています。

   このファイルを作成するには、次のコマンドを入力します。

   ```
   vim amazon-cloudwatch-agent.json
   ```

   このファイルには、次の内容を追加します。

   ```
   {
     "traces": {
       "traces_collected": {
         "app_signals": {}
       }
     },
     "logs": {
       "metrics_collected": {
         "app_signals": {}
       }
     }
   ```

```
}
```

3. CloudWatchAgentServerPolicy と AWSXrayWriteOnlyAccess の各 IAM ポリシーを Amazon EC2 インスタンスの IAM ロールにアタッチします。

   a. AWS Management Console にサインインして、IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。

   b. [ロール] を選択し、Amazon EC2 インスタンスに使用しているロールを検索します。次に、そのロール名を選択します。

   c. [アクセス権限] タブで、[アクセス権限の追加]、[ポリシーのアタッチ] の順に選択します。

   d. CloudWatchAgentServerPolicy を検索します。必要に応じて検索ボックスを使用してください。次に、そのポリシーのチェックボックスをオンにして、[アクセス権限の追加] を選択します。

   e. AWSXrayWriteOnlyAccess を検索します。必要に応じて検索ボックスを使用してください。次に、そのポリシーのチェックボックスをオンにして、[アクセス権限の追加] を選択します。

4. 次のコマンドを入力して CloudWatch エージェントを起動します。*agent-config-file-path* は CloudWatch エージェント設定ファイルへのパス (`./amazon-cloudwatch-agent.json` など) に置き換えます。例に示すように、`file:` プレフィックスも記述する必要があります。

```
export CONFIG_FILE_PATH=./amazon-cloudwatch-agent.json
```

```
sudo /opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-ctl \
-a fetch-config \
-m ec2 -s -c file:$CONFIG_FILE_PATH
```

ステップ 3: 計測を設定しアプリケーションを起動する

次のステップは、CloudWatch Application Signals のアプリケーションを計測することです。

Java

Amazon EC2 インスタンスで Application Signals を有効にする手順内で、Java アプリケーションを計測可能にするには

1. AWS Distro for OpenTelemetry Java 自動計測エージェントの最新バージョンをダウンロードします。このリンクを使用すると、最新バージョンをダウンロードできます。リリース済みバージョンをすべて確認するには、aws-otel-java-instrumentation のリリースを参照してください。
2. Application Signals の利点を最適化するには、環境変数によって情報を追加した後にアプリケーションを起動します。この情報は、Application Signals のダッシュボードに表示されます。

   a. `OTEL_RESOURCE_ATTRIBUTES` 変数には、次の情報をキーと値のペアとして指定します。

      - アプリケーションの稼働環境を設定するには、`aws.hostedin.environment` を使用します。これは、アプリケーションの [ホスト元] 環境として Application Signals ダッシュボードに表示されます。この属性キーは Application Signals にのみ使用され、X-Ray トレースのアノテーションと CloudWatch メトリクスのディメンションに変換されます。このキーの値を指定しない場合、`Generic` (デフォルト値) が使用されます。
      - サービス名を設定するには、`service.name` を使用します。これは、Application Signals ダッシュボードにアプリケーションのサービス名として表示されます。このキーの値を指定しない場合、`unknown_service` (デフォルト値) が使用されます。

   b. `OTEL_EXPORTER_OTLP_TRACES_ENDPOINT` 変数には、トレースのエクスポート先となるベースエンドポイント URL を指定します。CloudWatch エージェントでは 4315 が OLTP ポートとして公開されています。Amazon EC2 では、アプリケーションがローカル CloudWatch エージェントと通信するため、この値を `OTEL_EXPORTER_OTLP_TRACES_ENDPOINT=http://localhost:4315` に設定する必要があります。

   c. `OTEL_AWS_APP_SIGNALS_EXPORTER_ENDPOINT` 変数には、メトリクスのエクスポート先となるベースエンドポイント URL を指定します。CloudWatch エージェントでは 4315 が OLTP ポートとして公開されています。Amazon EC2 では、アプリケーションがローカル CloudWatch エージェントと通信するため、この値を `OTEL_AWS_APP_SIGNALS_EXPORTER_ENDPOINT=http://localhost:4315` に設定する必要があります。

d. JAVA_TOOL_OPTIONS 変数には、AWS Distro for OpenTelemetry Java 自動計測エージェントが保存されているパスを指定します。

```
export JAVA_TOOL_OPTIONS=' -javaagent:$ADOT_AGENT_PATH'
```

例:

```
export ADOT_AGENT_PATH=./aws-opentelemetry-agent.jar
```

e. OTEL_METRICS_EXPORTER 変数の値は none に設定することをお勧めします。これにより、他のメトリクスエクスポーターが無効になり、Application Signals エクスポーターのみ使用されるようになります。

f. OTEL_AWS_APP_SIGNALS_ENABLED 変数については、OTEL_AWS_APP_SIGNALS_ENABLED を true に設定し、SpanMetricProcessor (SMP) を有効にします。これにより、トレースから、Application Signals のメトリクスが生成されます。

3. 前のステップで説明した環境変数を使用してアプリケーションを起動します。起動スクリプトの例を次に示します。

```
JAVA_TOOL_OPTIONS=' -javaagent:$ADOT_AGENT_PATH' \
OTEL_METRICS_EXPORTER=none \
OTEL_AWS_APP_SIGNALS_ENABLED=true \
OTEL_AWS_APP_SIGNALS_EXPORTER_ENDPOINT=http://localhost:4315 \
OTEL_EXPORTER_OTLP_TRACES_ENDPOINT=http://localhost:4315 \
OTEL_RESOURCE_ATTRIBUTES=aws.hostedin.environment=$YOUR_HOST_ENV,service.name=
$YOUR_SVC_NAME \
java -jar $MY_JAVA_APP.jar
```

Python

Amazon EC2 インスタンスで Application Signals を有効にする手順内で、Python アプリケーションを計測可能にするには

1. AWS Distro for OpenTelemetry Python 自動計測エージェントの最新バージョンをダウンロードします。次の コマンドを使用してインストールします。

```
pip install aws-opentelemetry-distro
```

リリース済みバージョンをすべて確認するには、「AWS Distro for OpenTelemetry Python instrumentation」を使用します。

2. Application Signals の利点を最適化するには、環境変数によって情報を追加した後にアプリケーションを起動します。この情報は、Application Signals のダッシュボードに表示されます。

   a. OTEL_RESOURCE_ATTRIBUTES 変数には、次の情報をキーと値のペアとして指定します。

      - アプリケーションの稼働環境を設定するには、aws.hostedin.environment を使用します。これは、アプリケーションの [ホスト元] 環境として Application Signals ダッシュボードに表示されます。この属性キーは Application Signals にのみ使用され、X-Ray トレースのアノテーションと CloudWatch メトリクスのディメンションに変換されます。このキーの値を指定しない場合、Generic (デフォルト値) が使用されます。
      - サービス名を設定するには、service.name を使用します。これは、Application Signals ダッシュボードにアプリケーションのサービス名として表示されます。このキーの値を指定しない場合、unknown_service (デフォルト値) が使用されます。

   b. OTEL_EXPORTER_OTLP_PROTOCOL 変数には、テレメトリデータを HTTP 経由で次のステップに記載されている CloudWatch エージェントエンドポイントにエクスポートする http/protobuf を指定します。

   c. OTEL_EXPORTER_OTLP_TRACES_ENDPOINT 変数には、トレースのエクスポート先となるベースエンドポイント URL を指定します。CloudWatch エージェントでは 4316 が HTTP 経由の OLTP ポートとして公開されています。Amazon EC2 では、アプリケーションがローカル CloudWatch エージェントと通信するため、この値を OTEL_EXPORTER_OTLP_TRACES_ENDPOINT=http://localhost:4316/v1/traces に設定する必要があります。

   d. OTEL_AWS_APP_SIGNALS_EXPORTER_ENDPOINT 変数には、メトリクスのエクスポート先となるベースエンドポイント URL を指定します。CloudWatch エージェントでは 4316 が HTTP 経由の OLTP ポートとして公開されています。Amazon EC2 では、アプリケーションがローカル CloudWatch エージェントと通信するため、この値を OTEL_AWS_APP_SIGNALS_EXPORTER_ENDPOINT=http://localhost:4316/v1/metrics に設定する必要があります。

e. OTEL_METRICS_EXPORTER 変数の値は none に設定することをお勧めします。これにより、他のメトリクスエクスポーターが無効になり、Application Signals エクスポーターのみ使用されるようになります。

f. OTEL_AWS_APP_SIGNALS_ENABLED 変数については、OTEL_AWS_APP_SIGNALS_ENABLED を true に設定し、SpanMetricProcessor を有効にします。これにより、トレースから、Application Signals のメトリクスが生成されます。

3. 前のステップで説明した環境変数を使用してアプリケーションを起動します。起動スクリプトの例を次に示します。

- $HOST_ENV をアプリケーションが稼働しているホスト環境に置き換えてください。これは、アプリケーションの [ホスト元] 環境として Application Signals ダッシュボードに表示されます。
- $SVC_NAME をアプリケーションの名前に置き換えます。これは、Application Signals ダッシュボードにアプリケーション名として表示されます。
- $PYTHON_APP をアプリケーションの場所と名前に置き換えます。

```
OTEL_METRICS_EXPORTER=none \
OTEL_LOGS_EXPORTER=none \
OTEL_AWS_APP_SIGNALS_ENABLED=true \
OTEL_PYTHON_DISTRO=aws_distro \
OTEL_PYTHON_CONFIGURATOR=aws_configurator \
OTEL_EXPORTER_OTLP_PROTOCOL=http/protobuf \
OTEL_TRACES_SAMPLER=xray \
OTEL_TRACES_SAMPLER_ARG="endpoint=http://localhost:2000" \
OTEL_AWS_APP_SIGNALS_EXPORTER_ENDPOINT=http://localhost:4316/v1/metrics \
OTEL_EXPORTER_OTLP_TRACES_ENDPOINT=http://localhost:4316/v1/traces \
OTEL_RESOURCE_ATTRIBUTES=aws.hostedin.environment=$HOST_ENV,service.name=
$SVC_NAME \
opentelemetry-instrument python $PYTHON_APP.py
```

Python アプリケーションの Application Signals を有効にする前に、以下の考慮事項に注意してください。

- コンテナ化されたアプリケーションの一部は、PYTHONPATH 環境変数がないことが原因でアプリケーションが起動しなくなることがあります。これを解決するには、PYTHONPATH 環境変数をアプリケーションの作業ディレクトリの場所に設定します。

これは OpenTelemetry の自動計測に関する既知の問題によるものです。この問題の詳細については、「Python autoinstrumentation setting of PYTHONPATH is not compliant」を参照してください。

- Django アプリケーションには、OpenTelemetry Python ドキュメントで概説されている追加の必須設定があります。
  - `--noreload` フラグを使用すると、自動リロードを防ぐことができます。
  - Django アプリケーションの `settings.py` ファイルの場所に `DJANGO_SETTINGS_MODULE` 環境変数を設定します。これにより、OpenTelemetry がユーザーの Django 設定に正しくアクセスして統合できるようになります。

Application Signals のインストールでトラブルシューティングを行う

⚠ Application Signals はプレビューリリース中です。この機能についてご意見がありましたら、app-signals-feedback@amazon.com までご連絡ください。

このセクションでは、CloudWatch Application Signals のトラブルシューティングに役立つヒントを紹介します。

トピック

Application Signals を有効にしたが、その後、アプリケーションを起動できない

Application Signals を Amazon EKS クラスターのアプリケーションで有効にした後にそのアプリケーションを起動できない場合は、次の点を確認してください。

- アプリケーションが別のモニタリングソリューションによって計測されていないかを確認します。Application Signals は、他の計測ソリューションと共存できません。

- アプリケーションが Application Signals を使用するための互換性要件を満たしていることを確認します。詳細については、「Application Signals のサポート対象システム」を参照してください。
- アプリケーションで AWS Distro for OpenTelemetery Java または Python エージェントや CloudWatch エージェントイメージなどの Application Signals アーティファクトを取得できなかった場合は、ネットワークの問題である可能性があります。

この問題を軽減するには、アプリケーションデプロイマニフェストからアノテーション `instrumentation.opentelemetry.io/inject-java: "true"` または `instrumentation.opentelemetry.io/inject-python: "true"` を削除し、アプリケーションを再デプロイします。その後、アプリケーションが動作するかどうかを確認します。

Application Signals を有効にしたが、その後、Python アプリケーションを起動できない

これは、PYTHONPATH 環境変数がないとアプリケーションが起動しないことがあるという OpenTelemetry 自動計測の既知の問題です。これを解決するには、PYTHONPATH 環境変数をアプリケーションの作業ディレクトリの場所に設定してください。この問題の詳細については、「PPython autoinstrumentation setting of PYTHONPATH is not compliant with Python's module resolution behavior, breaking Django applications」を参照してください。

Django アプリケーションには、OpenTelemetry Python ドキュメントで概説されている追加の必須設定があります。

- `--noreload` フラグを使用すると、自動リロードを防ぐことができます。
- Django アプリケーションの `settings.py` ファイルの場所に `DJANGO_SETTINGS_MODULE` 環境変数を設定します。これにより、OpenTelemetry がユーザーの Django 設定に正しくアクセスして統合できるようになります。

CloudWatch と X-Ray でテレメトリデータを確認できない

Application Signals のダッシュボードでメトリクスまたはトレースを確認できない場合は、次の原因が考えられます。こうした原因の調査は、最終更新の後、Application Signals によるデータの収集と表示が完了するまで 15 分ほど待った後に開始してください。

- ADOT Java エージェントが、使用しているライブラリとフレームワークに対応していることを確認してください。詳細については、「Libraries / Frameworks」を参照してください。

- CloudWatch エージェントが実行中であることを確認します。最初に、CloudWatch エージェントポッドのステータスを確認し、いずれのポッドも Running のステータスであることを確認します。

  ```
  kubectl -n amazon-cloudwatch get pods.
  ```

  CloudWatch エージェント設定ファイルに次の記述を追加してデバッグログを有効にし、エージェントを再起動します。

  ```
  "agent": {
  >>>>>>> streams
    "region": "${REGION}",
    "debug": true
  },
  ```

  その後、CloudWatch エージェントポッドでエラーが発生していないか確認します。

- CloudWatch エージェントの設定に問題がないか確認します。CloudWatch エージェント設定ファイルに次の記述があり、その記述を追加した後にエージェントを再起動したことを確認します。

  ```
  "agent": {
    "region": "${REGION}",
    "debug": true
  },
  ```

  その後、OpenTelemetry のデバッグログに次のようなエラーメッセージがないか確認します: `ERROR io.opentelemetry.exporter.internal.grpc.OkHttpGrpcExporter - Failed to export ...`。こうしたメッセージは、問題を示している可能性があります。

  それでも問題が解決しない場合は、`kubectl describe pod` コマンドでポッドを記述し、ダンプの後、`OTEL_` で始まる名前の環境変数を確認します。

- OpenTelemetry Python デバッグログを有効にするには、環境変数 `OTEL_PYTHON_LOG_LEVEL` を debug に設定して、アプリケーションを再デプロイします。

- CloudWatch エージェントからデータをエクスポートするためのアクセス権限の付与が間違っていないかや、十分であるかを確認します。CloudWatch エージェントのログに `Access Denied` のメッセージが見られる場合は、その点が問題の可能性があります。例えば、CloudWatch エージェントのインストール時に適用したアクセス権限が後で変更されたり削除されたりしたかもしれません。

- テレメトリデータの生成中に AWS Distro for OpenTelemetry (ADOT) の問題が生じていないかを確認します。

  計測のアノテーションである `instrumentation.opentelemetry.io/inject-java` と `sidecar.opentelemetry.io/inject-java` のそれぞれがアプリケーションのデプロイメントに適用され、値が `true` に設定されていることを確認してください。これらが適切でない場合、ADOT アドオンを正しくインストールしていても、アプリケーションポッドは計測されません。

  次に、`Init` コンテナがアプリケーションに適用され、`Ready` 状態が `True` であることを確認します。`init` コンテナが Ready でない場合は、ステータスを確認してその理由を特定してください。

  問題が解決しない場合は、次の手順で OpenTelemetry Java SDK のデバッグログを有効にして、`ERROR io.telemetry` で始まるメッセージを検索します。

  デバッグログを有効にするには、環境変数 `OTEL_JAVAAGENT_DEBUG` を true に設定して、アプリケーションを再デプロイします。
- メトリクスまたはスパンエクスポーターの処理でデータの一部が送信されなかった可能性があります。原因を特定するには、アプリケーションログで「`Failed to export...`」が含まれるメッセージを確認してください。
- CloudWatch エージェントによってメトリクスまたはスパンが Application Signals に送信される際に、同エージェントがスロットリングされた可能性があります。スロットリングを示すメッセージを CloudWatch エージェントのログで確認してください。

依存関係メトリクスに不明な値がある

Application Signals ダッシュボードの依存関係名またはオペレーションに UnknownOperation、UnknownRemoteService、UnknownRemoteOperation が表示される場合は、不明なリモートサービスやリモートオペレーションに関するデータの発生とそれらのデプロイに一致が見られるかを確認してください。これは、Application Signals の既知の問題であり、今後のリリースで修正される予定です。

Amazon CloudWatch Observability EKS アドオンの管理で生じた ConfigurationConflict への対処

Amazon CloudWatch Observability EKS アドオンのインストールまたは更新時に、タイプが `ConfigurationConflict` の `Health Issue` によって生じたエラー (`Conflicts found when`

`trying to apply. Will not continue due to resolve conflicts mode` で始まる) が見られたとします。これはおそらく、CloudWatch エージェントとその関連コンポーネント (ServiceAccount、ClusterRole、ClusterRoleBinding など) がクラスターにインストールされていることが原因と考えられます。このアドオンによって CloudWatch エージェントとその関連コンポーネントがインストールされるときに内容の変更が検出されると、デフォルトではインストールまたは更新が失敗します。これによって、クラスター上にあるリソースの状態が上書きされることを防いでいます。

Amazon CloudWatch Observability EKS アドオンへのオンボード時にこのエラーが発生した場合は、クラスターにインストール済みの CloudWatch エージェントセットアップを削除した後に EKS アドオンをインストールすると良いでしょう。カスタムエージェント設定など、元の CloudWatch エージェントセットアップへのカスタマイズを必ずバックアップし、Amazon CloudWatch Observability EKS アドオンを次回インストールまたは更新するときにそれらのカスタマイズを適用してください。Container Insights へのオンボーディングを目的に CloudWatch エージェントをインストール済みである場合は、「Container Insights の CloudWatch エージェントと Fluent Bit の削除」で詳細を確認してください。

その他に、アドオンでサポートされている設定オプションで `OVERWRITE` を指定して競合を解決することも可能です。このオプションを使用すると、クラスター上の競合を上書きしてアドオンのインストールまたは更新を継続できます。Amazon EKS コンソールを使用する場合、アドオンの作成または更新時に [オプションの構成設定] を選択すると、[競合の解決方法] が表示されます。AWS CLI を使用する場合は、コマンドに `--resolve-conflicts OVERWRITE` を指定することで、アドオンを作成または更新できます。

Application Signals を設定する

> ⚠ Application Signals はプレビューリリース中です。この機能についてご意見がありましたら、app-signals-feedback@amazon.com までご連絡ください。

このセクションでは、CloudWatch Application Signals の設定について説明します。

トレースのサンプリングレート

デフォルトの場合、有効化後の Application Signals では、デフォルトのサンプリングレート設定 (`reservoir=1/s` と `fixed_rate=5%`) を使用して、一元化された X-Ray サンプリングを実行できるようになります。AWS Distro for OpenTelemetry (ADOT) SDK エージェントの環境変数は次のように設定されています。

環境変数	Value	注記
OTEL_TRACES_SAMPLER	xray	
OTEL_TRACES_SAMPLER_ARG	endpoint=http://cloudwatch-agent.amazon-cloudwatch:2000	CloudWatch エージェントのエンドポイント

サンプリング設定の変更方法については、以下を参照してください。

- X-Ray のサンプリングを変更するには、「サンプリングルールのカスタマイズ」を参照してください。
- ADOT のサンプリングを変更するには、「Configuring the OpenTelemetry Collector for X-Ray remote sampling」を参照してください。

一元化された X-Ray サンプリングを無効にしてローカルサンプリングを使用する場合は、ADOT SDK Java エージェントに次の値を指定します。この例では、サンプリングレートが 5% に設定されます。

環境変数	Value
OTEL_TRACES_SAMPLER	parentbased_traceidratio
OTEL_TRACES_SAMPLER_ARG	0.05

高度なサンプリング設定については、「OTEL_TRACES_SAMPLER」を参照してください。

高カーディナリティ操作の管理

Application Signals には、操作のカーディナリティを管理したり、コストを最適化するためにメトリクスのエクスポートを管理したりできる CloudWatch エージェントの設定が含まれています。デフォルトでは、あるサービスの個別操作の数がデフォルトのしきい値である 500 件を超えると、メトリクス制限機能が有効になります。構成設定を調整することで動作を調整できます。

メトリクス制限が有効かどうかを確認する

以下の方法を使用して、デフォルトのメトリクス制限が行われているかどうかを確認できます。行われている場合は、次のセクションのステップに従ってカーディナリティの制御の最適化を検討する必要があります。

- CloudWatch コンソールで、[Application Signals]、[サービス] の順に選択します。[AllOtherOperations] という名前の [操作]、または [AllOtherRemoteOperations] という名前の [RemoteOperation] が表示される場合は、メトリクス制限が行われています。
- Application Signals によって収集されたメトリクスのいずれかに `Operation` ディメンションの値「`AllOtherOperations`」がある場合、メトリクス制限が行われています。
- Application Signals によって収集されたメトリクスのいずれかに `RemoteOperation` ディメンションの値「`AllOtherRemoteOperations`」がある場合、メトリクス制限が行われています。

カーディナリティコントロールの最適化

カーディナリティコントロールを最適化するには、以下を実行できます。

- カスタムルールを作成して操作を集約します。
- メトリクス制限ポリシーを設定します。

カスタムルールを作成して操作を集約する

高カーディナリティ操作は、コンテキストから抽出された不適切な一意の値が原因で発生することがあります。例えば、ユーザー ID やセッション ID をパスに含む HTTP/S リクエストを送信すると、何百もの個別の操作が発生する可能性があります。このような問題を解決するには、CloudWatch エージェントにカスタマイズルールを設定して、これらの操作を書き換えることをお勧めします。

`PUT /api/customer/owners/123`、`PUT /api/customer/owners/456` などの個別の `RemoteOperation` 呼び出しで多数の異なるメトリクスの生成が急増している場合は、これらの操作を単一の `RemoteOperation` に統合することをおすすめします。1 つのアプローチは、`PUT /api/customer/owners/` で始まるすべての `RemoteOperation` 呼び出しを統一された形式 (具体的には `PUT /api/customer/owners/{ownerId}`) に標準化することです。以下に示しているのはこのポリシーの例です。その他のカスタマイズルールについては、「CloudWatch Application Signals を有効にする」を参照してください。

```
{
   "logs":{
```

```
      "metrics_collected":{
         "app_signals":{
            "rules":[
               {
                  "selectors":[
                     {
                        "dimension":"RemoteOperation",
                        "match":"PUT /api/customer/owners/*"
                     }
                  ],
                  "replacements":[
                     {
                        "target_dimension":"RemoteOperation",
                        "value":"PUT /api/customer/owners/{ownerId}"
                     }
                  ],
                  "action":"replace"
               }
            ]
         }
      }
   }
}
```

また、カーディナリティの高いメトリクスが AllOtherRemoteOperations に集約されていて、具体的にどのようなメトリクスが含まれているのか不明瞭な場合もあります。CloudWatch エージェントは、ドロップされた操作を記録できます。ドロップされた操作を特定するには、次の例の設定を使用して、問題が再発するまでログ記録を有効にします。次に、CloudWatch エージェントログ (コンテナ stdout または EC2 ログファイルからアクセス可能) を調べ、キーワード drop metric data を検索します。

```
{
  "agent": {
    "config": {
      "agent": {
        "debug": true
      },
      "traces": {
        "traces_collected": {
          "app_signals": {
          }
        }
```

```
      },
      "logs": {
        "metrics_collected": {
          "app_signals": {
            "limiter": {
              "log_dropped_metrics": true
            }
          }
        }
      }
    }
  }
}
```

メトリクス制限ポリシーを作成する

デフォルトのメトリクス制限設定がサービスのカーディナリティに対応していない場合は、メトリクス制限設定をカスタマイズできます。これを行うには、CloudWatch エージェント設定ファイルの `logs/metrics_collected/app_signals` セクション内に `limiter` セクションを追加します。

次の例では、メトリクス制限のしきい値を 500 個の個別メトリクスから 100 個に引き下げています。

```
{
  "logs": {
    "metrics_collected": {
      "app_signals": {
        "limiter": {
          "drop_threshold": 100
        }
      }
    }
  }
}
```

サービスレベル目標 (SLO)

⚠ Application Signals はプレビューリリース中です。この機能についてご意見がありましたら、app-signals-feedback@amazon.com までご連絡ください。

Application Signals を使用すると、重要な事業運営向けサービスに配慮したサービスレベル目標を作成できます。こうしたサービスに SLO を作成したら、SLO ダッシュボードでそれらを追跡できる上、最も重要なオペレーションをひとめで確認することも可能です。

また、重要なオペレーションの現在のステータスを表示できるようオペレーター向けにクイックビューを作成するだけでなく、SLO を使用してサービスの長期的なパフォーマンスを追跡したり、サービスが想定どおりに提供されているかどうかを確認したりすることも可能です。顧客とサービスレベル契約を結んでいる場合、SLO はそれを確実に満たす手段と言えるでしょう。

SLO を使用してサービスの正常性を評価するには、まず、主要なパフォーマンス指標 (サービスレベル指標 (SLI)) に基づいて、明確で測定可能な目標を設定します。SLO を導入すると、設定したしきい値と目標に照らして SLI パフォーマンスを追跡すると同時に、アプリケーションのパフォーマンス値としきい値との差がどの程度開いているかを報告できます。

Application Signals を使用すると、主要なパフォーマンス指標に SLO を設定できます。Application Signals では、検出したすべてのサービスとオペレーションについて `Latency` と `Availability` のメトリクスを自動的に収集します。多くの場合、こうしたメトリクスは、SLI としての使用に理想的であり、SLO 作成ウィザードに従うことで独自の SLO に利用できます。その後、Application Signals ダッシュボードから、全 SLO のステータスを追跡することが可能です。

SLO は、サービスで呼び出したり使用したりする特定のオペレーションに設定できます。また、`Latency` と `Availability` のメトリクスの他に、任意の CloudWatch メトリクスまたはメトリクス式を SLI として使用することも可能です。

CloudWatch Application Signals を最大限に活用するには、SLO の作成が非常に重要です。SLO を作成すると、Application Signals コンソールで重要なサービスやオペレーションのステータスを確認し、パフォーマンスが良いものや動作が異常なものを特定できます。SLO の追跡により、次のような大きな利点を得られます。

- SLI に照らして測定している重要なサービスについて、オペレーションの現在の正常性を簡単に確認できるため、問題のあるサービスやオペレーションを迅速に優先順位付けし、特定することが可能です。
- サービスのパフォーマンスを、測定可能なビジネス目標に照らして長期的に追跡できます。

SLO の設定対象を選択することで情報の重要度を指定でき、Application Signals ダッシュボードには重要度が高いと見なした情報が自動的に表示されます。

SLO の作成時には、CloudWatch アラームを作成して、SLO をモニタリングすることも可能です。しきい値越えや警告レベルをモニタリングするようアラームを設定できます。こうしたアラームに

よって、SLO メトリクスが設定したしきい値を超えたり、警告のしきい値に近づいたりした場合に自動的に通知を受けられます。例えば、SLO の値が警告のしきい値に近づくと通知が届くため、長期的なパフォーマンス目標を達成するには、そのアプリケーションでのチャーン抑制が必要であると気付くことができます。

トピック

SLO の概念

SLO の構成要素を次に示します。

- サービスレベル指標 (SLI): 指定対象のパフォーマンスメトリクスであり、アプリケーションに必要なパフォーマンスレベルを表します。Application Signals では、検出したサービスとオペレーションについて、主要なメトリクスである `Latency` と `Availability` を自動的に収集します。これらは、SLO の設定に理想的なメトリクスとなることが少なくありません。

  SLI には、使用するしきい値を選択します。例えば、レイテンシーには 200 ミリ秒などと指定します。
- 目標または達成度目標: 時間間隔のそれぞれで SLI のしきい値を満たすことが求められる時間の長さをパーセンテージで示した値です。この間隔には、数時間のように短い期間や、1 年のように長い期間を設定できます。

  カレンダー間隔とローリング間隔のいずれも設定可能です。
  - カレンダー間隔: カレンダーに従った間隔であり、1 か月ごとに追跡する SLO などがこれに該当します。CloudWatch では、1 か月の日数に基づいて、正常性、予算、達成度の数値を自動的に調整します。カレンダー間隔は、カレンダーに沿って測定するビジネス目標に適しています。
  - ローリング間隔: ローリングベースで測定を行います。この間隔は、アプリケーションで最近提供されたユーザーエクスペリエンスの追跡に適しています。
- 期間: 短い時間を意味し、間隔は、多数の期間がつながったものと言えます。間隔内の各期間でアプリケーションのパフォーマンスを SLI と比較し、期間ごとに、必要なアプリケーションパフォーマンスが達成されたかどうかを判断します。

例えば、99% という目標で、カレンダー間隔に 1 日、期間に 1 分が指定されている場合、その日のアプリケーションのパフォーマンスは、1 分の 99% に相当する期間、成功のしきい値に達していなければなりません。達していれば、その日の SLO を満たしたことになります。翌日は新規の評価間隔と見なされ、2 日目の SLO を満たすには、2 日目における 1 分の 99% に相当する期間、成功のしきい値に達していることが求められます。

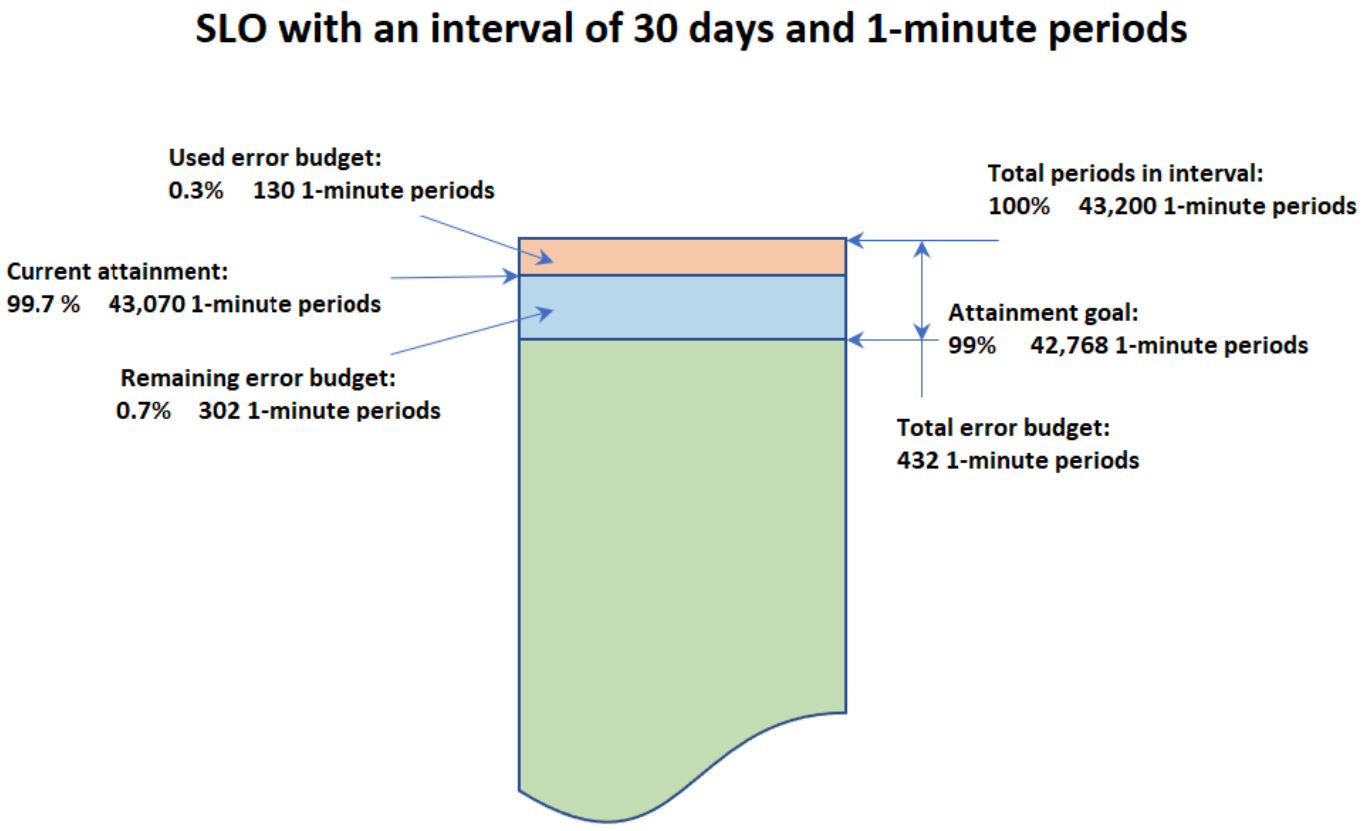
SLI には、Application Signals で収集した新規の標準アプリケーションメトリクスの 1 つ、あるいは、任意の CloudWatch メトリクスまたはメトリクス式を使用できます。標準アプリケーションメトリクスとして SLI に使用できるのは、`Latency` と `Availability` です。`Availability` は、成功の応答をリクエストの合計で割った数値で表し、(1 - 障害率)*100 のように計算します。ここでの Fault 応答数は `5xx` エラーの件数を意味し、成功の応答とは `5XX` エラーのない応答を指します。`4XX` 応答は成功の応答と見なされます。

> **Note**
>
> 現在、期間ベースの計算のみサポートされています。ボリュームベースまたはリクエストベースの計算は、今後のリリースでサポート対象となる予定です。

エラーバジェットと達成度を計算する

SLO の情報を表示すると、SLO の現在の正常性とその「エラーバジェット」を確認できます。エラーバジェットとは、対象の間隔内で、しきい値を超えても SLO を満たすことのできる時間の長さを意味し、「総エラーバジェット」は、対象の間隔全体で違反が許容される時間を合計したものです。「残りのエラーバジェット」は、現在の間隔内で許容される残りの違反時間であり、総エラーバジェットから既に発生した違反時間を差し引いた値で表します。

次の図は、間隔が 30 日、期間が 1 分、達成度目標が 99% の目標について、達成度とエラーバジェットの概念を示しています。30 日を分数で表すと 43,200 分であり、43,200 の 99% は 42,768 であるため、SLO を満たすには、当月のうち 42,768 分は正常性を確保しなければなりません。現在の間隔では、異常な期間 (1 分間) がこれまでに 130 回ありました。



SLO の達成状況を各期間内で判断する

SLI データは、各期間内で SLI に使用する統計に基づいて 1 つのデータポイントに集計され、このデータポイントが期間全体の長さを表すことになります。その 1 つのデータポイントが SLI のしきい値と比較され、その期間の正常性が判断されます。現在の時間範囲内で異常な期間をダッシュボードで確認できるため、サービスの優先順位付けが必要な状況にすぐに気付くことが可能です。

異常と判断された期間は、エラーバジェット上、その期間全体が違反期間としてカウントされます。エラーバジェットの追跡により、対象のサービスが必要なパフォーマンスを長期的に達成しているかどうかを把握できます。

SLO を作成する

重要なアプリケーションにはレイテンシーと可用性の両方の SLO を設定することをお勧めします。Application Signals で収集するこれらのメトリクスは一般的なビジネス目標として使用できます。

また、任意の CloudWatch メトリクスや、1 つの時系列となるメトリクス数学式に SLO を設定することもできます。

アカウントに SLO を初めて作成したときに CloudWatch がアカウントに AWSServiceRoleForCloudWatchApplicationSignals サービスにリンクされたロールをまだ作成していない場合は自動的に作成します。このサービスにリンクされたロールにより、CloudWatch は CloudWatch Logs データ、X-Ray トレースデータ、CloudWatch メトリクスデータ、タグ付けデータをアカウントのアプリケーションから収集できるようになります。CloudWatch サービスにリンクされたロールの詳細については、「CloudWatch のサービスにリンクされたロールの使用」を参照してください。

SLO を作成するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで、[サービスレベル目標 (SLO)] を選択します。
3. [SLO の作成] を選択します。
4. SLO の名前を入力します。サービスやオペレーションの名前と、レイテンシーや可用性といった適切なキーワードを含めると、優先順位付けの対象となっている SLO のステータスを簡単に特定できます。
5. [サービスレベルインジケータ (SLI) の設定] では、次のいずれかの操作を行います。
   - 標準アプリケーションメトリクスの `Latency` または `Availability` のいずれかに SLO を設定するには、次の操作を行います。
     a. [サービスオペレーション] を選択します。
     b. この SLO でモニタリングするサービスを選択します。
     c. この SLO でモニタリングするオペレーションを選択します。

        [サービスの選択] と [オペレーションの選択] ドロップダウンには、過去 24 時間以内にアクティブになったサービスとオペレーションが表示されます。
     d. [可用性] または [レイテンシー] を選択して、しきい値を設定します。
   - 任意の CloudWatch メトリクス、または CloudWatch メトリクスの数式に SLO を設定するには:
     a. [CloudWatch メトリクス] を選択します。
     b. [CloudWatch メトリクスの選択] を選択します。

        [メトリクスの選択] 画面が表示されます。[ブラウズ] タブまたは [クエリ] タブを使用して、対象のメトリクスを検索するか、メトリクスの数式を作成します。

対象のメトリクスを選択したら、[グラフ化したメトリクス] タブを選択し、SLO に使用する [統計] と [期間] を選択します。次に [Select metric] (メトリクスの選択) を選択します。

これらの画面の詳細については、「メトリクスをグラフ化する」と「CloudWatch グラフに数式を追加する」を参照してください。

c. [条件の設定] では、SLO で成功の指標として使用する比較演算子としきい値を選択します。

6. ステップ 5 で [サービスオペレーション] を選択した場合は、[詳細設定] を選択して、この SLO の期間を調整することもできます。

7. SLO の [間隔] と [達成度目標] を設定します。間隔と達成度目標、およびこれら 2 つの関連性については、「SLO の概念」を参照してください。

8. (オプション) SLO に CloudWatch のアラームまたは警告上のしきい値を 1 つ以上設定します。

   a. CloudWatch アラームを使用すると、SLI のパフォーマンスに基づいてアプリケーションの異常を判断し、Amazon SNS による通知をプロアクティブに受けることができます。

   アラームを作成するには、アラームのチェックボックスの 1 つを選択し、アラームが ALARM 状態になった旨の通知に使用する Amazon SNS トピックを入力または作成します。CloudWatch アラームの詳細については、「Amazon CloudWatch でのアラームの使用」を参照してください。アラームの作成には料金がかかります。CloudWatch の料金の詳細については、Amazon CloudWatch の料金をご覧ください。

   b. 警告のしきい値を設定すると、Application Signals の画面にその値が表示されるため、現在正常値であっても達成が難しくなりそうな SLO を特定できます。

   警告しきい値を設定するには、[警告しきい値] にしきい値を入力します。SLO のエラーバジェットが警告しきい値を下回ると、複数の Application Signals 画面でその SLO に [警告] のマークが付きます。警告しきい値は、エラーバジェットのグラフにも表示でき、警告しきい値に基づいて [SLO 警告アラーム] を作成することも可能です。

9. この SLO にタグを追加するには、[タグ] タブを選択し、[新規タグの追加] を選択します。タグは、リソースの管理、識別、整理、検索、フィルタリングに役立ちます。タグ付けの詳細については、「AWS リソースのタグ付け」を参照してください。

> **Note**
>
> この SLO に関連するアプリケーションが AWS Service Catalog AppRegistry に登録されている場合は、`awsApplication` タグを使用することで、その SLO を AppRegistry 内の対象アプリケーションに関連付けることができます。詳細については、「What is AppRegistry?」を参照してください。

10. [SLO の作成] を選択します。1 つ以上のアラームを作成すると、それに応じてボタン名が変更されます。

SLO ステータスの表示と優先順位付けを行う

CloudWatch コンソールの [サービスレベル目標] または [サービス] のオプションを使用すると、SLO の正常性をすぐに確認できます。[サービス] ビューでは、設定済み SLO を基に計算された異常なサービスの割合をひとめで把握できます。[サービス] オプションの使用については、「Application Signals を使用したアプリケーションの運用状態のモニタリング」で詳しく確認できます。

[サービスレベル目標] ビューでは、組織をマクロ的に確認できます。例えば、達成できている SLO と達成できていない SLO を全体的に把握することが可能です。これにより、期待どおりの成果を長期的に得られているサービスおよびオペレーションの数を、選択した SLI に基づいて確認できます。

[Service Level Objects] ビューですべての SLO を表示するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで、[サービスレベル目標 (SLO)] を選択します。

   [サービスレベル目標 (SLO)] リストが表示されます。

   SLO の現在のステータスは [SLI ステータス] 列ですぐに確認できます。すべての異常な SLO がリストの一番上に表示されるように並べ替えるには、異常な SLO がすべて一番上になるまで [SLI ステータス] 列を選択します。

   SLO テーブルには次のようなデフォルトの列があります。表示する列を調整するには、リストの上にある歯車アイコンを選択してください。目標、SLI、達成度、間隔の詳細については、「SLO の概念」を参照してください。

   - SLO の名前。

- [目標] 列には、各間隔における期間の割合が表示されます。この割合を SLO 目標の SLI しきい値に照らして達成しなければなりません。この列には、SLO の間隔の長さも表示されます。
- [SLI ステータス] には、アプリケーションの現在のオペレーション状態が正常かどうかが表示されます。現在選択している時間範囲内のいずれかの期間で SLO 違反が生じた場合、[SLI ステータス] に [異常] と表示されます。
- [終了時の達成度] は、選択した時間範囲の終了時点で達成したレベルを表します。この列を基準にして並べ替えると、達成できない可能性が最も高い SLO を確認できます。
- [達成度差] は、選択した時間範囲の開始時と終了時における達成レベルの差であり、これがマイナスの場合、メトリクスが下降傾向にあることを示しています。この列を基準にして並べ替えると、最近の SLO の傾向を確認できます。
- [終了時エラーバジェット (%)] は、異常な期間があっても SLO を達成したと見なされる期間の合計時間をパーセンテージで示した値です。これを 5% に設定した場合、SLI が異常であっても、その長さが間隔内の残りの期間で 5% 以下であれば SLO を達成したと見なされます。
- [エラーバジェット差] は、選択した時間範囲の開始時と終了時におけるエラーバジェットの差であり、これがマイナスの場合、メトリクスが下降傾向にあることを示しています。
- [終了時エラーバジェット (時間)] は、対象の間隔において、異常が発生しても SLO を達成したと見なされる時間の長さです。例えば、これが 14 分の場合、残りの間隔で SLI が異常な時間があっても、それが 14 分未満であれば、SLO を達成したと見なされます。
- [サービス]、[オペレーション]、[タイプ] 列には、この SLO が設定されているサービスとオペレーションの情報が表示されます。

3. SLO の達成度とエラーバジェットのグラフを表示するには、SLO 名の横にあるラジオボタンを選択します。

   ページ上部に表示されるそれらのグラフで、[SLO の達成度] と [エラーバジェット] のステータスを確認でき、この SLO に関連する SLI メトリクスのグラフも表示することが可能です。

4. 達成されていない SLO にさらに優先順位付けを行うには、その SLO に関連するサービス名またはオペレーション名を選択します。優先順位付けをさらに行える詳細ページが表示されます。詳細については、「サービスの詳細ページで詳細なサービスアクティビティとオペレーションヘルスを表示する」を参照してください。
5. そのページにあるチャートと表の時間範囲を変更するには、画面の上部近くで新しい時間範囲を選択します。

既存の SLO を編集する

既存の SLO を編集するには、次のステップに従います。SLO の編集で、変更できるのは、しきい値、間隔、達成度目標、タグのみです。それ以外のサービス、オペレーション、メトリクスなどを変更するには、既存の SLO を編集せず、SLO を新規作成します。

SLO で最も重要な要素 (期間やしきい値など) を変更すると、達成度や正常性に関する以前のデータポイントや評価がすべて無効になります。SLO は実質的に削除され、再作成されます。

> **Note**
>
> SLO を編集しても、その SLO に関連するアラームは、自動更新されません。SLO との同期を維持するには、アラームの更新が必要となる場合もあります。

既存の SLO を編集するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで、[サービスレベル目標 (SLO)] を選択します。
3. 編集する SLO の横にあるラジオボタンを選択し、[アクション]、[SLO の編集] を選択します。
4. 変更後、[変更の保存] を選択します。

SLO を削除する

既存の SLO を削除するには、次のステップに従います。

> **Note**
>
> SLO を削除しても、その SLO に関連するアラームは、自動削除されないため、手動で削除する必要があります。詳細については、「アラームの管理」を参照してください。

SLO を削除するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで、[サービスレベル目標 (SLO)] を選択します。
3. 編集する SLO の横にあるラジオボタンを選択し、[アクション]、[SLO の削除] を選択します。
4. [確認] を選択します。

Application Signals を使用したアプリケーションの運用状態のモニタリング

⚠ Application Signals はプレビューリリース中です。この機能についてご意見がありましたら、app-signals-feedback@amazon.com までご連絡ください。

CloudWatch コンソール内で Application Signals を使用すると、アプリケーションの運用状態のモニタリングとトラブルシューティングを行うことができます。

- アプリケーションサービスのモニタリング — 日常の運用状態のモニタリングの一環として、[サービス] ページを使用すると、すべてのサービスの概要を確認できます。障害率またはレイテンシーが最も高いサービスを確認したり、どのサービスのサービスレベル指標 (SLI) が正常でないかを確認したりできます。サービスを選択して [サービスの詳細] ページを開くと、詳細なメトリクス、サービスオペレーション、Synthetics Canary、およびクライアントリクエストを確認できます。これは、運用上の問題のトラブルシューティングと根本原因の特定に役立ちます。
- アプリケーションのトポロジの検査 — [サービスマップ] を使用すると、クライアント、Synthetics Canary、サービス、依存関係の間の関係を含むアプリケーションのトポロジを経時的に把握し、モニタリングすることができます。サービスレベル指標 (SLI) の状態を瞬時に確認したり、コール量、障害率、レイテンシーなどの主要な指標を確認したりできます。ドリルダウンすると、[サービスの詳細] ページで詳細情報を確認できます。

これらのページを使用することで、最初の問題検出から根本原因の特定に至るまでの運用サービスの状態の問題を迅速にトラブルシューティングできるシナリオの例をご覧ください。

Application Signals によって運用状態のモニタリングが可能になる仕組み

Application Signals でアプリケーションを有効にすると、アプリケーションサービス、API、およびそれらの依存関係が自動的に検出され、[サービス] ページ、[サービスの詳細] ページ、および [サービスマップ] ページに表示されます。Application Signals によって複数のソースから情報が収集されるので、サービスを検出したり、運用状態をモニタリングしたりできます。

- AWS Distro for OpenTelemetry (ADOT) — Application Signals を有効にすると、OpenTelemetry Java 自動インストルメンテーションライブラリは、CloudWatch エージェントによって収集されたメトリクスとトレースを出力するように設定されます。これらのメトリクスとトレースは、サービス、運用、依存関係、その他のサービス情報を検出するために使用されます。

- サービスレベル目標 (SLO) — サービスのサービスレベル目標を作成すると、[サービス] ページ、[サービスの詳細] ページ、[サービスマップ] ページにサービスレベル指標 (SLI) の状態が表示されます。SLI により、レイテンシー、可用性、その他の運用メトリクスをモニタリングできます。
- CloudWatch Synthetics Canary — Canary に X-Ray トレースを設定すると、Canary スクリプトからのサービスへの呼び出しがサービスに関連付けられ、[サービスの詳細] ページに表示されます。
- CloudWatch リアルユーザーモニタリング (RUM) — CloudWatch RUM ウェブクライアントで X-Ray トレースが有効になっている場合、サービスへのリクエストが自動的に関連付けられ、[サービスの詳細] ページに表示されます。
- AWS Service Catalog AppRegistry — Application Signals は、アカウント内の AWS リソースを自動検出し、これらのリソースを AppRegistry で作成された論理アプリケーションにグループ化できるようにします。[サービス] ページに表示されるアプリケーション名は、サービスが実行されている基盤のコンピュートリソースに基づいています。

> **Note**
>
> Application Signals は、ユーザーが選択した現在の時間フィルター内に出力されたメトリックとトレースに基づいて、サービスと運用を表示します。(デフォルトでは過去 3 時間です。) 現在の時間フィルター内にサービス、運用、依存関係、Synthetics Canary、またはクライアントページにアクティビティが発生しなかった場合は表示されません。
> 現時点では、最大 1,000 個のサービスを表示できます。サービスとサービストポロジの検出は最大で 10 分遅れることがあります。サービスレベル指標 (SLI) の状態の評価は最大で 15 分遅れることがあります。

サービスページで全体的なサービスアクティビティと運用状態を確認する

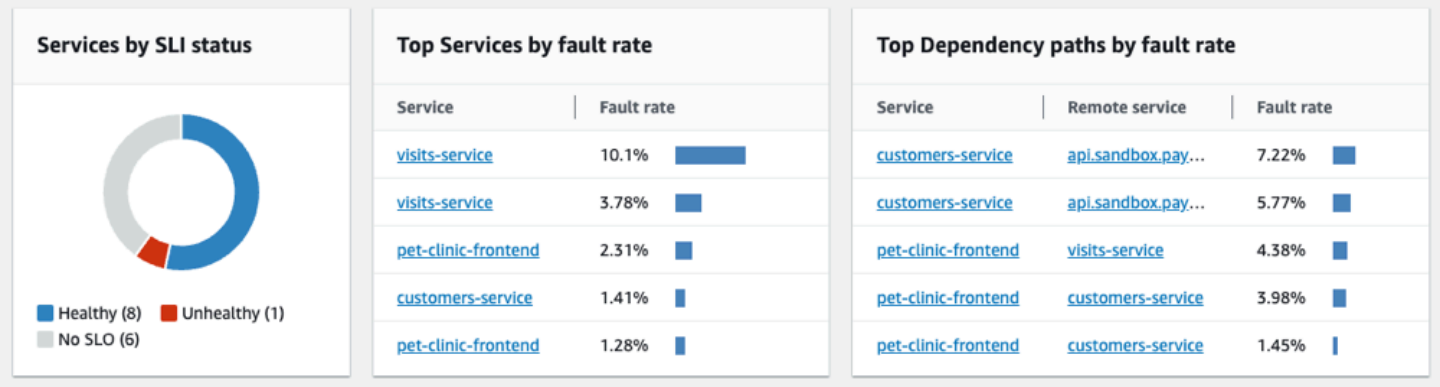
> **⚠** Application Signals は Amazon CloudWatch のプレビューリリースであり、変更される可能性があります。

サービスページでは、Application Signals が有効になっているサービスのリストを確認できます。また、オペレーションのメトリクスを表示して、どのサービスに異常なサービスレベル指標 (SLI) があるかをすばやく確認することもできます。ドリルダウンしてパフォーマンスの異常を探しながら、運用上の問題の根本原因を特定します。このページを表示するには、CloudWatch コンソールを開き、左側のナビゲーションペインの [Application Signals] セクションで [サービス] を選択します。

サービスの運用状態に関するメトリクスを詳しく見る

サービスページの上部には、サービス全体の運用状態のグラフと、上位のサービスとサービス依存関係を障害発生率別に表示した複数のテーブルが表示されます。左側のサービスグラフには、現在のページレベルの時間フィルターで正常または異常なサービスレベル指標 (SLI) が発生したサービスの数の内訳が表示されます。SLI により、レイテンシー、可用性、その他の運用メトリクスをモニタリングできます。

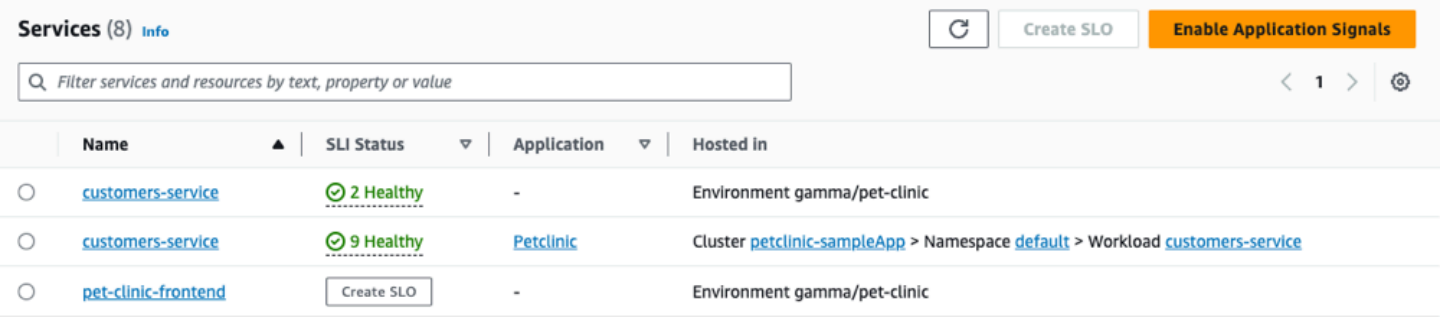
グラフの横にある 2 つのテーブルには、上位サービスのリストが障害発生率別に表示されます。いずれかのテーブルのサービス名を選択すると、サービスの詳細ページが開き、サービスオペレーションの詳細が詳しく表示されます。依存関係パスを選択して詳細ページを開き、サービス依存関係の詳細を表示します。ページの右上でより長い期間のフィルターを選択した場合でも、過去 3 時間までの情報が両方のテーブルに表示されます。



サービステーブルで運用状態を監視する

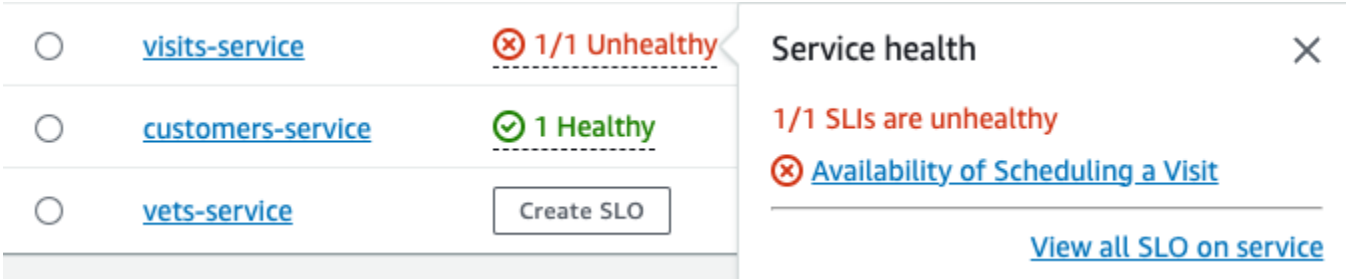
サービステーブルには、Application Signals が有効になっているサービスのリストが表示されます。[Application Signals を有効にする] を選択してセットアップページを開き、サービスの設定を開始します。詳細については、「Application Signals を有効にする」を参照してください。

フィルターテキストボックスから 1 つまたは複数のプロパティを選択して、サービステーブルをフィルタリングすると、探しているものを見つけやすくなります。各プロパティを選択すると、フィルター条件が表示されます。フィルターテキストボックスの下に、すべてのフィルターが表示されます。[フィルターのクリア] を選択すると、いつでもテーブルのフィルターを削除できます。



テーブル内のサービスの名前を選択すると、サービスレベルのメトリクス、オペレーション、その他の詳細を含むサービスの詳細ページが表示されます。サービスの基盤となるコンピュートリソースを AppRegistry のアプリケーションまたは AWS Management Consoleホームページのアプリケーションカードに関連付けている場合は、アプリケーション名を選択すると、myApplications コンソールページにアプリケーションの詳細が表示されます。Amazon EKS でホストされているサービスの場合は、[ホスト元] 列内の任意のリンクを選択すると、CloudWatch Container Insights 内のクラスター、名前空間、またはワークロードが表示されます。Amazon ECS または Amazon EC2 で実行されているサービスの場合は、環境値が表示されます。

サービスレベル指標 (SLI) のステータスは、サービスごとにテーブルに表示されます。サービスの SLI ステータスを選択すると、異常な SLI へのリンクと、そのサービスのすべての SLO を確認するためのリンクを含むポップアップが表示されます。

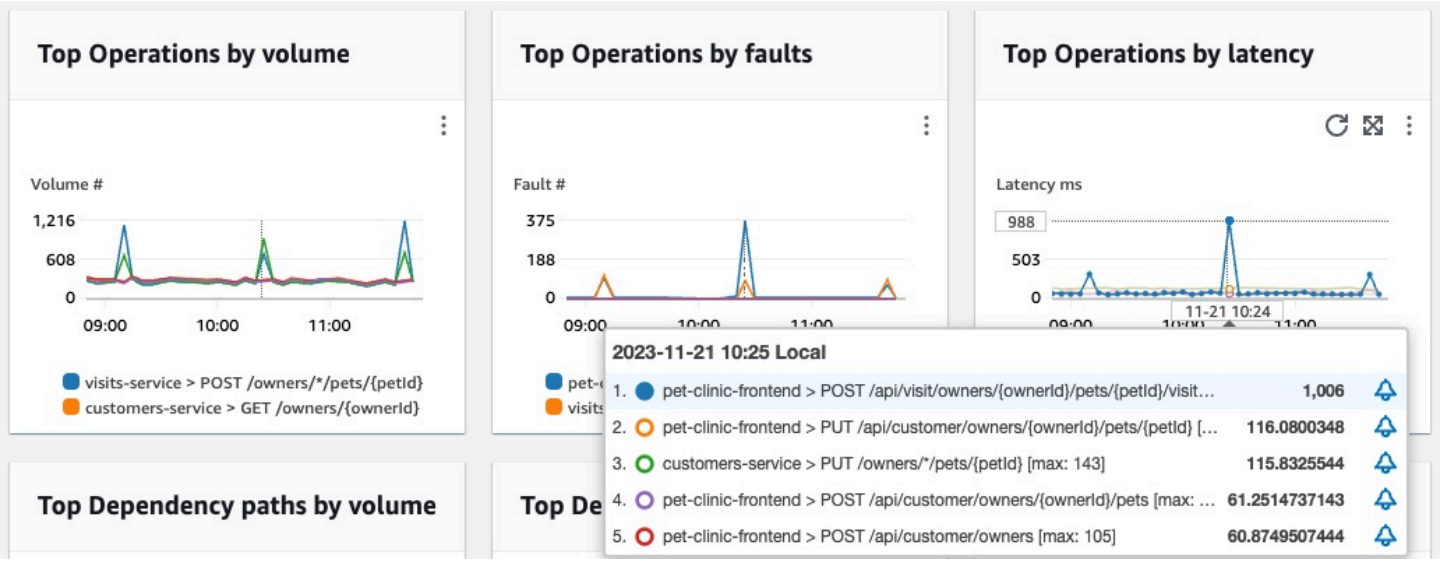


サービスの SLO が作成されていない場合は、[SLI ステータス] 列の [SLO の作成] ボタンを選択します。サービスに追加の SLO を作成するには、サービス名の横にあるオプションボタンを選択し、テーブルの右上にある [SLO の作成] を選択します。SLO を作成すると、どのサービスとオペレーションが正常に実行されていて、どれが異常かがひとめでわかります。詳細については、「service level objectives (SLOs)」を参照してください。

上位のオペレーションと依存関係のメトリクスを確認する

サービステーブルの下には、すべてのサービスにわたる上位のオペレーションと依存関係が、呼び出し量、障害、レイテンシーごとに表示されます。この一連のグラフには、すべてのサービスで異常が発生する可能性のあるオペレーションと依存関係に関する重要な情報が表示されます。グラフ内の任

意のポイントを選択すると、より詳細な一連の情報を含むポップアップが表示されます。グラフの下部にある一連の説明にカーソルを合わせると、特定の運用や依存関係パスに関する詳細なメトリクスを含むポップアップが表示されます。グラフの右上隅にあるコンテキストメニューボタンを選択すると、CloudWatch メトリクスやログページの表示などの追加オプションが表示されます。



サービスの詳細ページで詳細なサービスアクティビティとオペレーションヘルスを表示する

⚠ Application Signals は Amazon CloudWatch のプレビューリリースであり、変更される可能性があります。

アプリケーションを計測すると、Amazon CloudWatch Application Signals は、アプリケーションが検出したすべてのサービスをマッピングします。特定のサービスのサービス概要、オペレーション、依存関係、Canary、クライアントリクエストを確認するには、サービスの詳細ページを使用します。これらの詳細を表示するには、以下を実行します。

- CloudWatch コンソールを開きます。
- 左側のナビゲーションペインの [Application Signals] のセクションで[サービス] をクリックします。
- [サービス]、[上位サービス]、[依存関係] のテーブルから、任意のサービスの名前を選択します。

サービスの詳細ページは、次のタブで構成されています。

- [概要] - このタブは、オペレーション、依存関係、Synthetics、クライアントページの数など、特定のサービスの概要を表示するときに使用します。また、サービス全体に関する主要なメトリクス、上位のオペレーション、依存関係が表示されます。これらのメトリクスには、そのサービスのすべてのサービスオペレーションの、レイテンシー、障害、エラーに関する時系列データが含まれます。
- [サービスオペレーション] — このタブは、サービスが呼び出すオペレーションの一覧と、各オペレーションのヘルスを測定する主要メトリクスで構成された、実況グラフを表示するときに使用します。グラフ内のデータポイントを選択すると、そのデータポイントに関連付けられたトレース、ログ、メトリクスに関する情報を取得できます。
- [依存関係] — このタブは、サービスが呼び出す依存関係の一覧と、それら依存関係のメトリクスの一覧を表示するときに使用します。
- [Synthetics Canary] — このタブは、サービスへのユーザーコールをシミュレートする Synthetics Canary の一覧と、それら Canary の主要なパフォーマンスメトリクスを表示するときに使用します。
- [クライアントページ] — このタブは、サービスを呼び出すクライアントページの一覧と、クライアントとアプリケーションとのインタラクションの質を測定する、メトリクスを表示するときに使用します。

サービス概要の表示

サービス概要ページを使用すると、すべてのサービスオペレーションの、メトリクスの概要を 1 か所で表示できます。オペレーション、依存関係、クライアントページ、Synthetics Canary のすべてとアプリケーションとのインタラクションのパフォーマンスをチェックします。この情報があれば、問題を特定し、エラーをトラブルシューティングし、最適化の機会の見つけるために、どこに重点をおくべきかを判断するのに役立ちます。

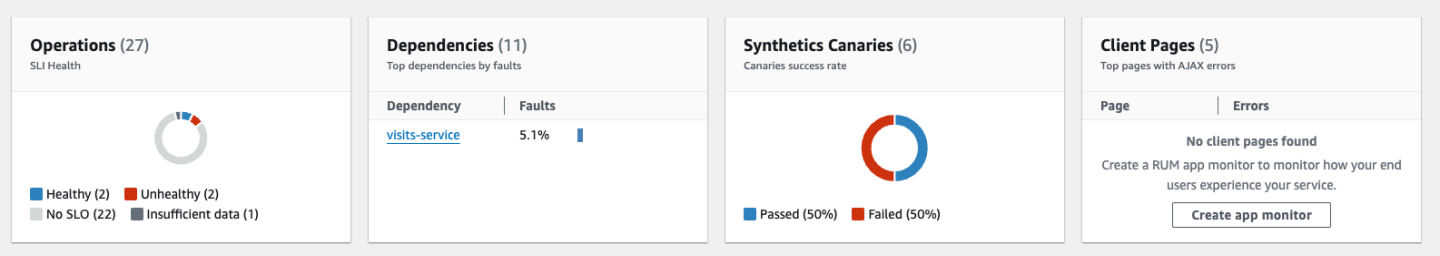
[サービスの詳細] で任意のリンクをクリックし、特定のサービスに関連する情報を表示します。例えば、Amazon EKS でホストされているサービスの場合、サービスの詳細ページに [クラスター]、[名前空間]、[ワークロード] の情報が表示されます。Amazon ECS または Amazon EC2 でホストされているサービスの場合、[サービスの詳細] ページに [環境] の値が表示されます。

[サービス] の [概要] タブには、以下の項目の概要が表示されます。

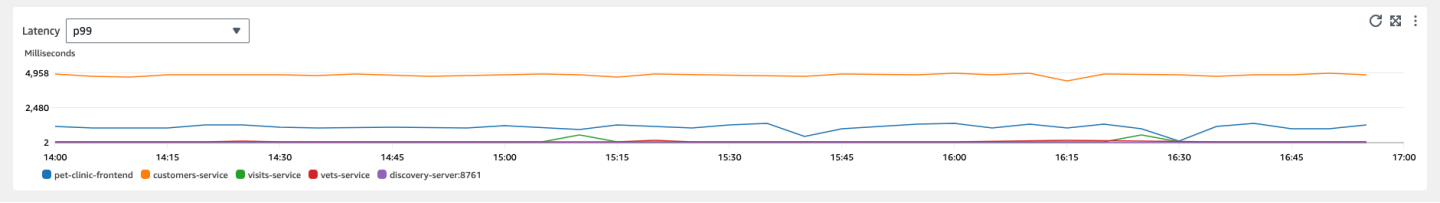
- [オペレーション] – このタブは、サービスオペレーションのヘルスを確認するときに使用します。ヘルスの状態は、サービスレベル目標 (SLO) の一部として定義されているサービスレベル指標 (SLI) によって判定されます。

- [依存関係] — このタブは、アプリケーションによって呼び出されるサービスの、上位の依存関係を障害率別に表示するときに使用します。
- [Synthetics Canary] – このタブは、サービスに関連付けられたエンドポイントまたは API への、シミュレートされた呼び出しの結果と、失敗した Canary の数を表示するときに使用します。
- [クライアントページ] – このタブは、非同期 JavaScript と XML (AJAX) にエラーがあるクライアントが呼び出した、トップページを表示するときに使用します。

次の図に、サービスの概要を示します。



[概要] タブには、すべてのサービスのうち、レイテンシーが最も高い上位 4 件の依存関係のグラフも表示されます。p99、p90、p50 の各レイテンシーメトリクスを使用すると、次のように、どの依存関係がサービス全体のレイテンシーの一因になっているかをすばやく評価できます。

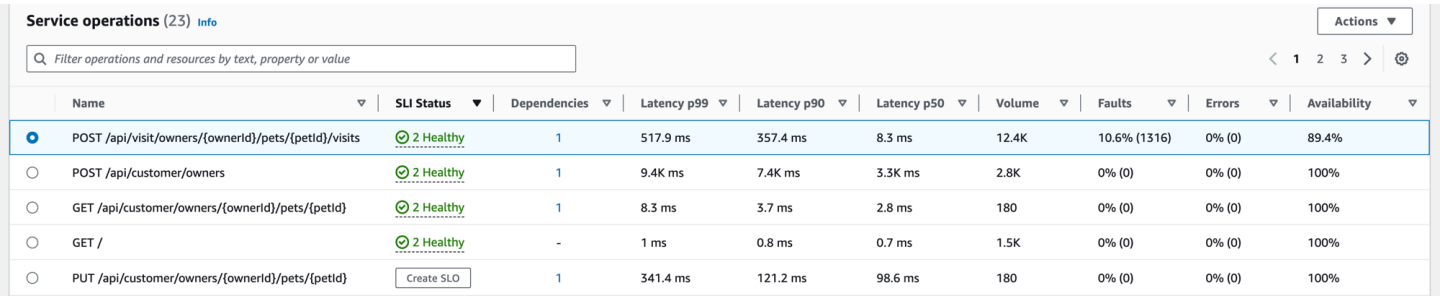


例えば、上記のグラフはカスタマーサービスの依存関係に対して行われたリクエストの 99% が完了までに約 4,950 ミリ秒かかっていることを示しています。他の依存関係はそこまで時間がかかっていません。

上位 4 つのサービスオペレーションをレイテンシー別に示したグラフでは、次のイメージのように、サービスごとにリクエスト量、可用性、障害率、エラー率が表示されます。

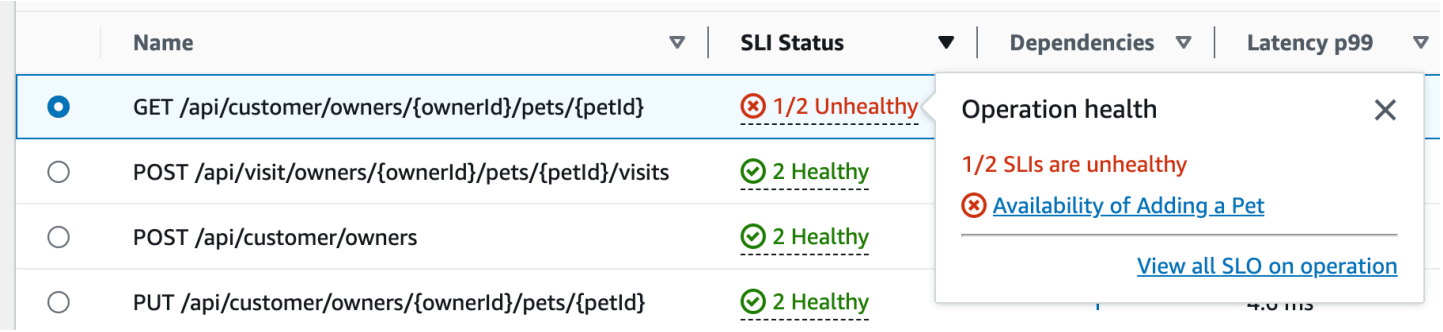
サービスオペレーションを表示する

アプリケーションを計測すると、Application Signals は、アプリケーションが呼び出したすべてのサービスオペレーションを検出します。[サービスオペレーション] タブは、サービスオペレーションを含むテーブルと、選択したオペレーションのパフォーマンスを測定する、一連のメトリクスを表示するときに使用します。このメトリクスには、次に示すように、SLI のステータス、依存関係の数、レイテンシー、ボリューム、可用性が含まれています。



フィルターテキストボックスから 1 つまたは複数のプロパティを選択してテーブルをフィルタリングすると、サービスオペレーションが見つけやすくなります。各プロパティを選択すると、フィルター条件が表示され、フィルターテキストボックスの下にフィルター全体が表示されます。[フィルターのクリア] を選択すると、いつでもテーブルのフィルターを削除できます。

オペレーションの SLI ステータスを選択すると、次のテーブルに示すように、異常な SLI へのリンクと、そのオペレーションのすべての SLO を確認するためのリンクがポップアップに表示されます。



サービスオペレーションテーブルには、SLI のステータス、正常または異常な SLI の数、各オペレーションの SLO の合計数、が一覧表示されます。

SLI は、サービスの運用状態を測定するレイテンシー、可用性、その他運用メトリクスをモニタリングするために使用されます。SLO は、サービスとオペレーションのパフォーマンスと運用状態をチェックするために使用されます。

SLO を作成するには、次の手順を実行します。

- オペレーションに SLO がない場合は、[SLI ステータス] 列で [SLO の作成] をクリックします。
- オペレーションに既に SLO がある場合は、次の手順を実行します。
  - オペレーション名の横にあるラジオボタンをクリックします。
  - テーブルの右上にある[アクション] の下矢印から [SLO の作成] を選択します。

詳細については、「サービスレベル目標 (SLO)」を参照してください。

[依存関係] 列には、このオペレーションが呼び出す依存関係の数が表示されます。この数を選択すると、選択したオペレーションにフィルタリングされた [依存関係] タブが開きます。

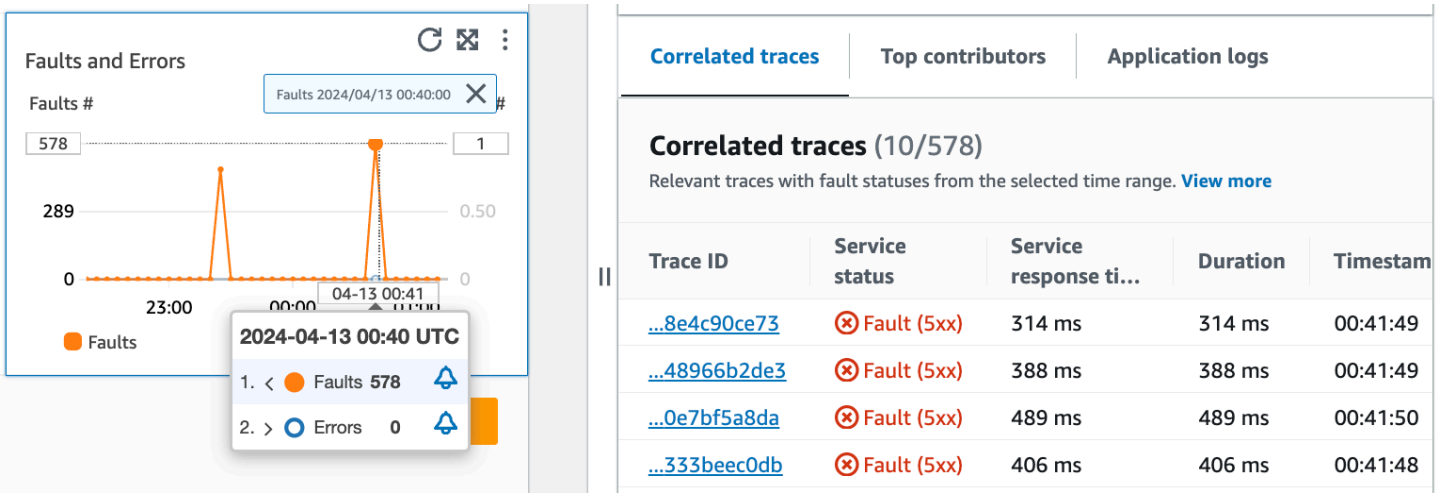
サービスオペレーションのメトリクス、相関トレース、およびアプリケーションログの表示

Application Signals は、サービスオペレーションメトリクスを AWS X-Ray トレース、CloudWatch Container Insights、アプリケーションログに関連付けます。これらのメトリクスを使用して、運用状態の問題をトラブルシューティングします。メトリクスをグラフで表示するには、次の手順を実行します。

1. [サービスオペレーション] のテーブルでサービスオペレーションを選択すると、[ボリュームと可用性]、[レイテンシー]、[障害とエラー] のメトリクスを示すテーブルの上に、選択したオペレーションの一連のグラフが表示されます
2. グラフ内のポイントにカーソルを合わせると、詳しい情報が表示されます。

3. グラフでポイントを選択すると、そのポイントの相関トレース、メトリクス、アプリケーションログを示す診断ペインが開きます。

次のイメージに示すように、グラフ内のポイントにカーソルを合わせるとツールヒントが表示され、ポイントをクリックすると診断ペインが表示されます。ツールヒントでは、[障害とエラー] グラフ内の対応するデータポイントに関する情報を確認できます。このペインには、選択されているポイントに関連する [相関トレース]、[上位の寄与要因]、[アプリケーションログ] が表示されます。



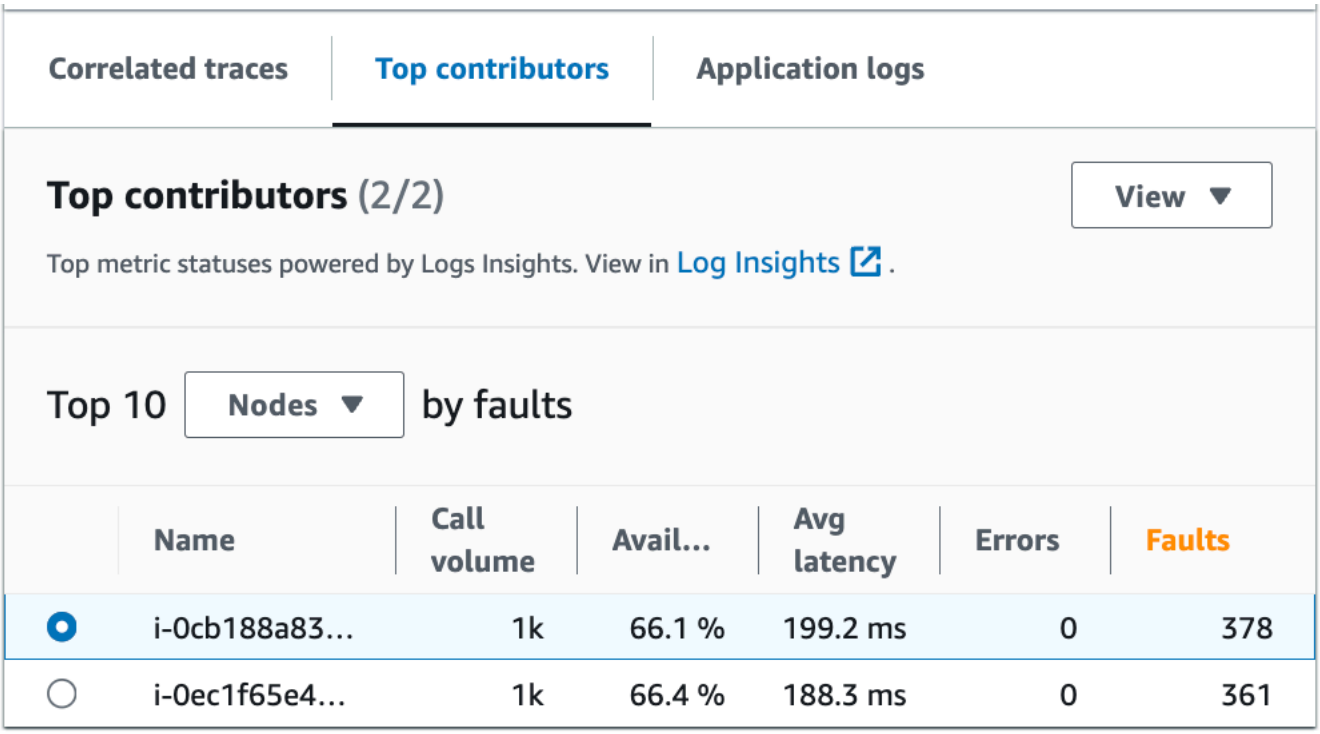
相関トレース

トレースの根本的な問題を理解するには、関連するトレースを確認します。相関するトレース、またはそれらに関連付けられたサービスノードが、類似した動作をしているかどうかを確認できます。相関するトレースを調べるには、[相関するトレース] のテーブルで [トレース ID] を選択し、選択したトレースの [X-Ray トレースの詳細] ページを開きます。このトレース詳細ページには、選択されているトレースに関連するサービスノードのマップと、トレースセグメントのタイムラインが表示されます。

上位の寄稿者

上位の寄稿者を表示して、メトリクスへの主要な入力ソースを特定します。寄稿者を、さまざまなコンポーネントごとにグループ化して、グループ内の類似点を特定し、トレース間での動作の違いを見きわめます。

[上位の寄稿者] タブに、各グループの [呼び出し量]、[可用性]、[平均レイテンシー]、[エラー]、[障害] のメトリクスが表示されます。次の例には、Amazon EKS プラットフォームにデプロイされたアプリケーションの、一連のメトリクスの上位の寄稿者が示されています。



上位の寄稿者には、次のメトリクスが含まれています。

- [呼び出し量] - 呼び出し量は、特定グループの、時間間隔あたりのリクエスト数を理解するときに使用します。
- [可用性] - 可用性は、特定のグループで、障害が検出されなかった時間の割合を把握するときに使用します。
- [平均レイテンシー] - レイテンシーは、調査対象のリクエストが実行された時期に応じて、特定のグループに対して特定の期間実行されたリクエストの、平均時間を確認するときに使用します。15 日以上前に行われたリクエストは、1 分間隔で評価されます。15 ~ 30 日前に行われたリクエストは、5 分間隔で評価されます。例えば、15 日前に障害の原因となったリクエストを調査している場合、呼び出し量メトリクスは 5 分間隔あたりのリクエスト数と等しくなります。
- [エラー] - 特定の時間間隔で測定されたグループあたりのエラーの数。
- [障害] - 特定の時間間隔におけるグループあたりの障害の数。

Amazon EKS または Kubernetes を使用している上位の寄稿者

Amazon EKS または Kubernetes にデプロイされたアプリケーションの、上位の寄稿者の情報を使用して、[ノード]、[ポッド]、[PodTemplateHash] でグループ化された運用状態のメトリクスを確認します。各要因の定義は次のとおりです。

- ポッドは、ストレージとリソースを共有する 1 つ以上の Docker コンテナから成るグループです。Kubernetes プラットフォームにデプロイできる最小単位がポッドです。ポッド別にグループ化すると、エラーがポッド固有の制限に関連しているかどうかを確認できます。
- ノードは、ポッドを実行するサーバーです。ノード別にグループ化すると、エラーがノード固有の制限に関連しているかどうかを確認できます。
- ポッドテンプレートハッシュは、特定のバージョンのデプロイを検出するために使用します。ポッドテンプレートハッシュでグループ化すると、エラーが特定のデプロイに関連しているかどうかを確認できます。

Amazon EC2 を使用している上位の寄稿者

Amazon EKS にデプロイされたアプリケーションの、上位の寄稿者の情報を使用して、[インスタンス ID] および [Auto Scaling グループ] でグループ化された運用状態のメトリクスを確認します。各要因の定義は次のとおりです。

- インスタンス ID は、サービスが実行されている Amazon EC2 インスタンスの一意の識別子です。インスタンス ID でグループ化すると、エラーが特定の Amazon EC2 インスタンスに関連しているかどうかを確認できます。
- Auto Scaling グループは、アプリケーションのリクエストに応える必要があるリソースをスケールアップするかスケールダウンする際に使用できる Amazon EC2 インスタンスの集まりです。必要に応じて Auto Scaling グループでグループ化すると、エラーの範囲がグループ内のインスタンスに限定されているかどうかを確認できます。

カスタムプラットフォームを使用している上位の寄稿者

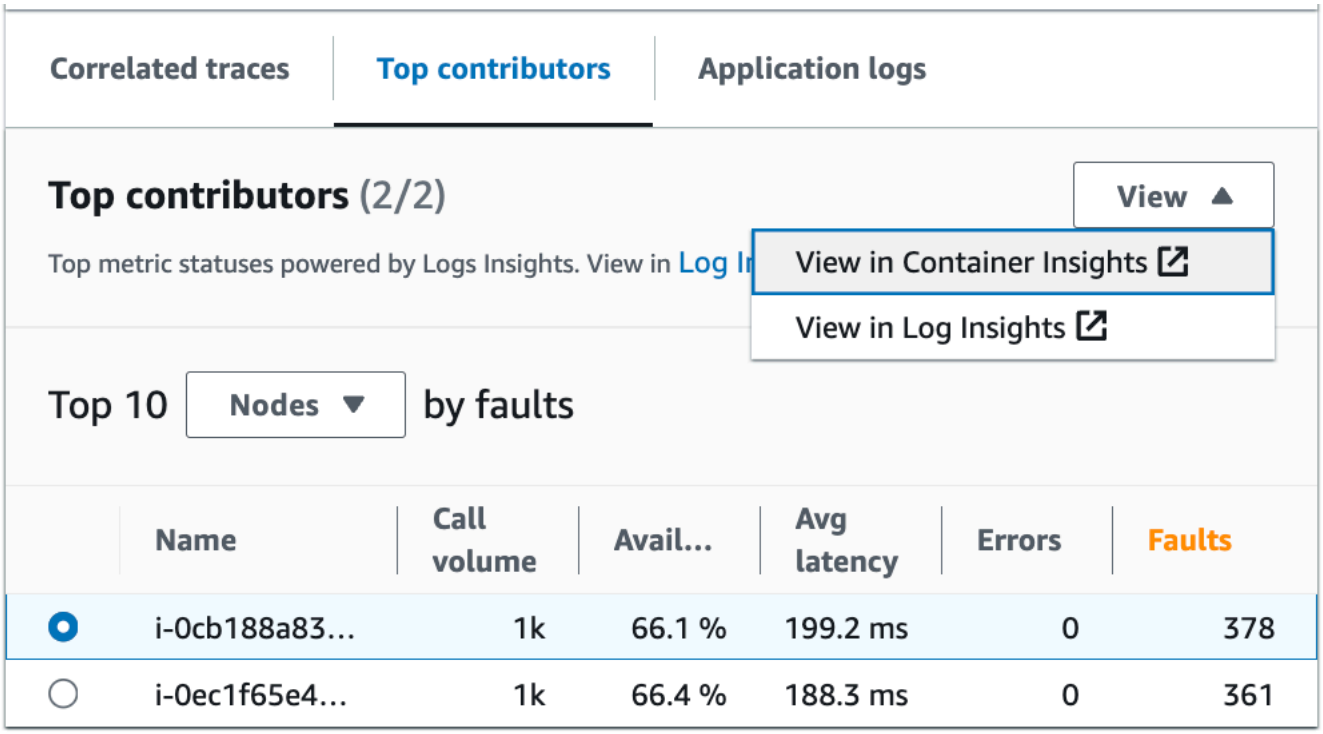
カスタム計測を使用してデプロイされたアプリケーションの、上位の寄稿者の情報を使用して、[ホスト名] でグループ化された運用状態のメトリクスを確認します。各要因の定義は次のとおりです。

- ホスト名は、ネットワークに接続されているエンドポイントや Amazon EC2 インスタンスなどのデバイスを識別するものです。ホスト名でグループ化すると、エラーが特定の物理デバイスまたは仮想デバイスに関連しているかどうかを確認できます。

Log Insights および Container Insights で上位の寄稿者を表示する

Log Insights で、上位の寄稿者に関するメトリクスを生成する自動クエリを表示し、修正します。Container Insights で、インフラストラクチャのパフォーマンスメトリクスを、ポッドやノードなど特定のグループごとに表示します。クラスター、ノード、またはワークロードをリソース消費量

別にソートして、エンドユーザーエクスペリエンスが影響を受ける前に異常をすばやく特定したり、プロアクティブにリスクを軽減したりできます。次のイメージに、これらのオプションを選択する方法を示します。



Container Insights では、Amazon EKS コンテナや Amazon ECS コンテナに関するメトリクスのうち、上位の寄与要因のグループ化に固有のものを表示できます。例えば、EKS コンテナをポッド別にグループ化して上位の寄与要因を生成した場合、Container Insights にはポッドでフィルタリングされたメトリクスと統計が表示されます。

Log Insights では、次の手順を使用して、[上位の寄与要因] にメトリクスを生成したクエリを変更できます。

1. [Log Insights に表示] を選択します。[Log Insights] ページが開いて、自動的に生成されたクエリが表示されます。次の情報が含まれています。

   - ログクラスターグループ名。
   - CloudWatch で調査していたオペレーション。
   - グラフで操作したオペレーションヘルスメトリクスの集計。

   ログ結果は自動的にフィルタリングされて、サービスグラフ上のデータポイントを選択するまでの過去 5 分間のデータが表示されます。

2. クエリを編集するには、生成されたテキストを変更後の内容に置き換えます。また、クエリジェネレーターを使用して、新しいクエリを生成したり、既存のクエリを更新したりすることもできます。

アプリケーションログ

[アプリケーションログ] タブのクエリを使用して、現在のロググループ、サービスに関するログ情報を生成し、タイムスタンプを挿入します。ロググループは、アプリケーションを設定する際に定義できるログストリームのグループです。

ロググループを使用して、次のような特性を持つログを整理します。

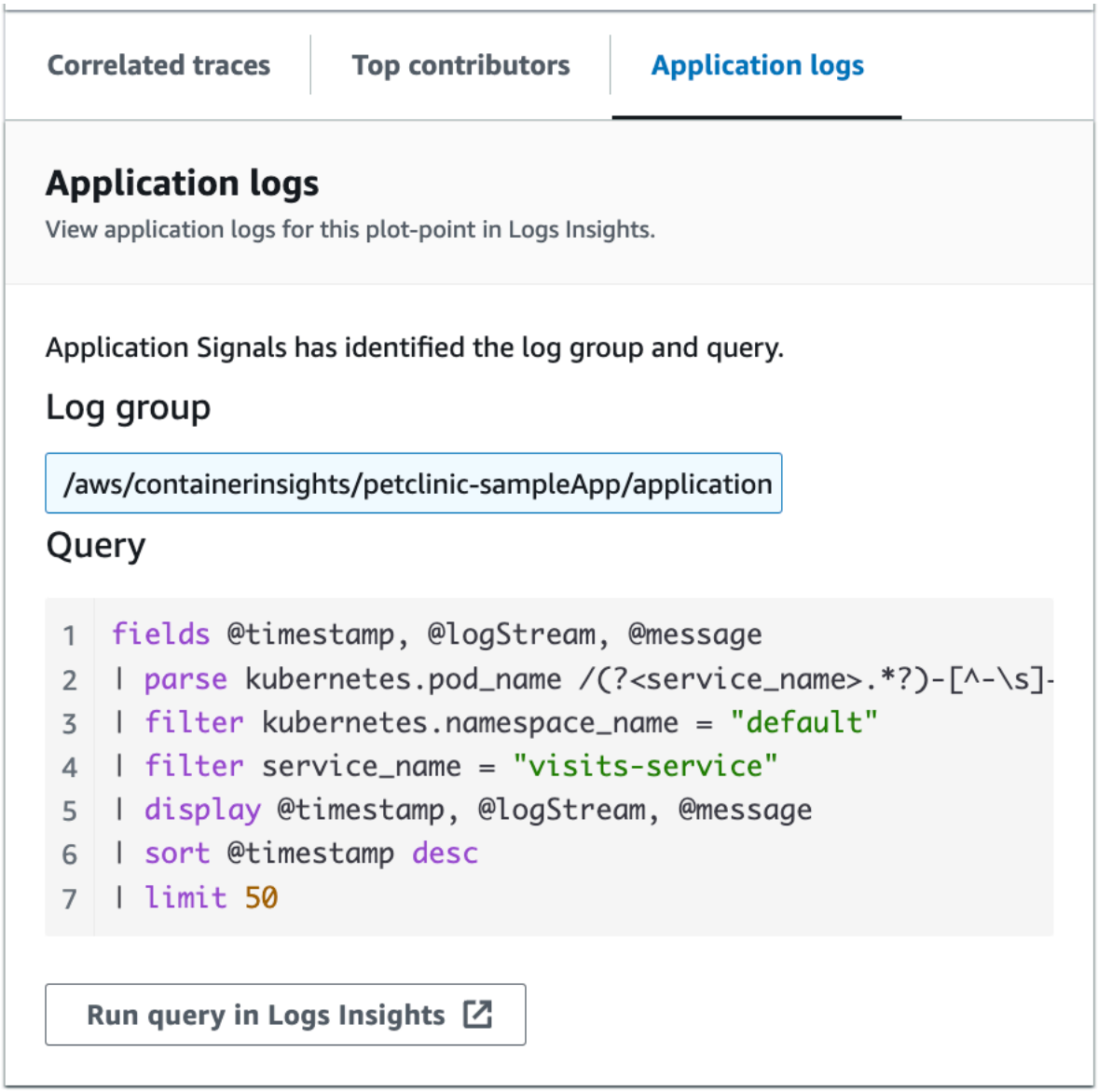
- 特定の組織、ソース、機能からログをキャプチャします。
- 特定のユーザーがアクセスするログをキャプチャします。
- 特定の期間のログをキャプチャします。

これらのログストリームを使用して、特定のグループまたは時間枠を追跡します。これらのロググループのモニタリングルール、アラーム、通知を設定することもできます。ロググループの詳細については、「ロググループとログストリームを操作」を参照してください。

アプリケーションログクエリは、ログ、繰り返し発生するテキストパターン、およびグラフィカルに可視化したロググループを返します。

クエリを実行するには、[Logs Insights でクエリを実行] を選択して、自動生成されたクエリを実行するか、クエリを変更します。クエリを編集するには、自動生成されたテキストを変更後の内容に置き換えます。また、クエリジェネレーターを使用して、新しいクエリを生成したり、既存のクエリを更新したりすることもできます。

次のイメージに、サービスオペレーショングラフで選択されているポイントに基づいて自動的に生成されたサンプルのクエリを示します。



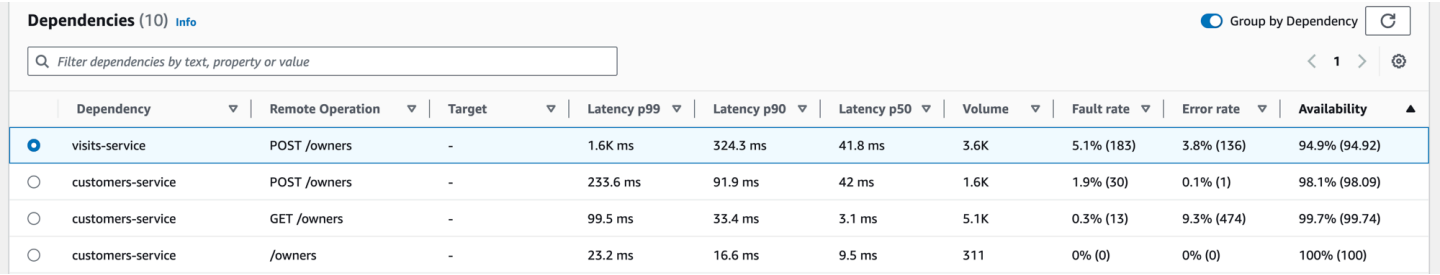
前のイメージでは、CloudWatch は選択されたポイントに関連付けられているロググループを自動的に検出して、生成されたクエリに含めていました。

サービスの依存関係を表示する

[依存関係] タブを選択すると、[依存関係] テーブルと、すべてのサービスオペレーションまたは 1 つのオペレーションの依存関係に関する一連のメトリクスが表示されます。このテーブルには、レイテンシー、呼び出し量、障害率、エラー率、可用性のメトリクスなど、Application Signals によって検出された依存関係のリストが含まれています。

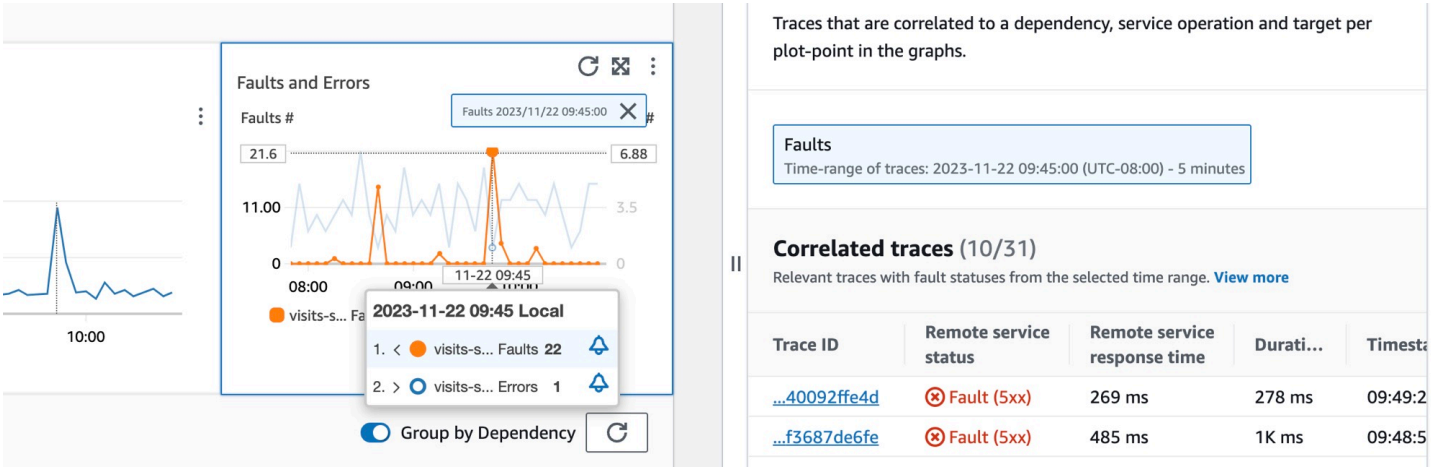
ページ上部の下向き矢印のリストからオペレーションを選択して依存関係を表示するか、[すべて] を選択してすべてのオペレーションの依存関係を表示します。

フィルターテキストボックスから 1 つまたは複数のプロパティを選択して、テーブルをフィルタリングすると、探しているものを見つけやすくなります。各プロパティを選択すると、フィルター条件が表示され、フィルターテキストボックスの下にフィルター全体が表示されます。[フィルターのクリア] を選択すると、いつでもテーブルのフィルターを削除できます。テーブルの右上にある [依存関係別にグループ化] を選択すると、依存関係をサービスとオペレーション名でグループ化できます。グループ化がオンになっている場合は、依存関係名の横にある [+] アイコンを使用して、依存関係のグループを展開するか折りたたみます。



[依存関係] 列には依存関係サービス名が表示され、[リモートオペレーション] 列にはサービスオペレーション名が表示されます。AWS のサービスを呼び出すと、[ターゲット] 列には DynamoDB のテーブルや Amazon SNS のキューといった AWS リソースが表示されます。

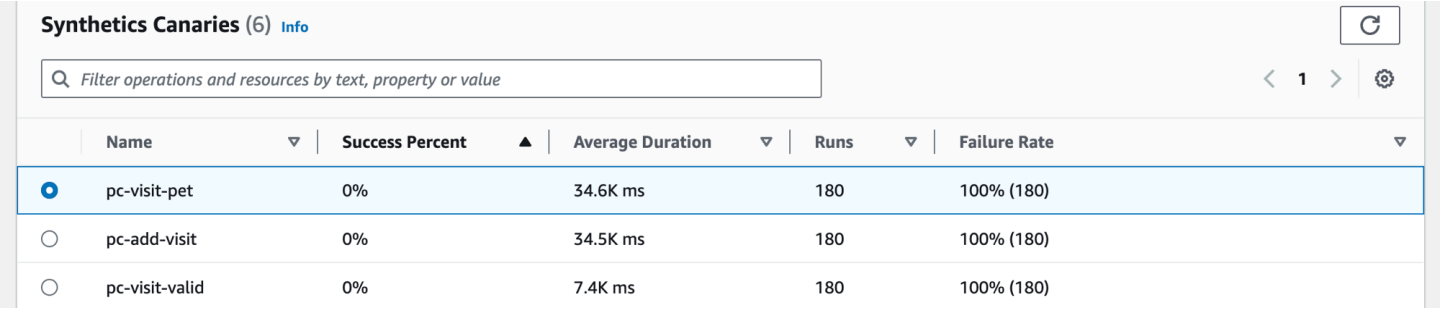
依存関係を選択するには、[依存関係] テーブルの依存関係の横にあるオプションを選択します。呼び出し量、可用性、障害、エラーに関する詳細なメトリクスを表示する一連のグラフが表示されます。グラフ内のポイントにカーソルを合わせると、ポップアップが開いて詳しい情報が表示されます。グラフ内のポイントを選択すると、診断ペインが開いて、グラフ内でのそのポイントの相関トレースが表示されます。[相関トレース] テーブルからトレース ID を選択して、選択したトレースの [X-Ray トレースの詳細] ページを開きます。



Synthetics Canary を表示する

[Synthetics Canary] タブを選択すると、[Synthetics Canary] テーブルが表示され、そのテーブルには一連のメトリクスが canary ごとに表示されます。このテーブルには、成功率、平均所要時間、実行、失敗率に関するメトリクスが含まれています。AWS X-Ray トレースが有効になっている Canary のみが表示されます。

Synthetics Canary テーブルのフィルターテキストボックスを使用して、関心のある Canary を見つけます。作成する各フィルターは、フィルターテキストボックスの下に表示されます。[フィルターのクリア] を選択すると、いつでもテーブルのフィルターを削除できます。



Canary の名前の横にあるラジオボタンをクリックすると、成功、エラー、期間など、グラフの詳細なメトリクスを含む一連のタブが表示されます。グラフ内のポイントにカーソルを合わせると、ポップアップが開いて詳しい情報が表示されます。グラフ内のポイントを選択すると、選択したポイントと相関する Canary 実行を示す、診断ペインが開きます。Canary 実行を選択して [ランタイム] をクリックすると、ログ、HTTP アーカイブ (HAR) ファイル、スクリーンショット、問題のトラブルシューティングに役立つ推奨のステップなど、選択した Canary 実行のアーティファクトが表示されます。[詳細] をクリックして、[Canary 実行] の横にある [CloudWatch Synthetics Canary] のページを開きます。

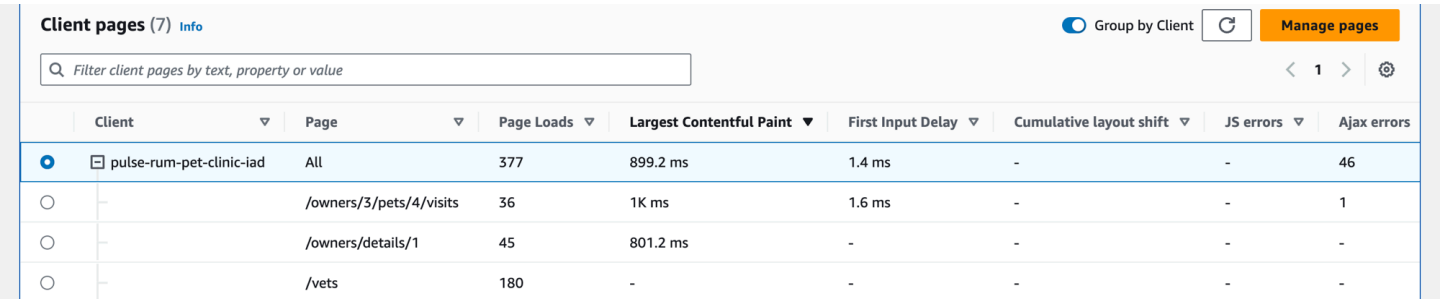
クライアントページを表示する

[クライアントページ] タブをクリックすると、サービスを呼び出すクライアントウェブページの一覧が表示されます。サービスまたはアプリケーションを操作する際のクライアントエクスペリエンスの

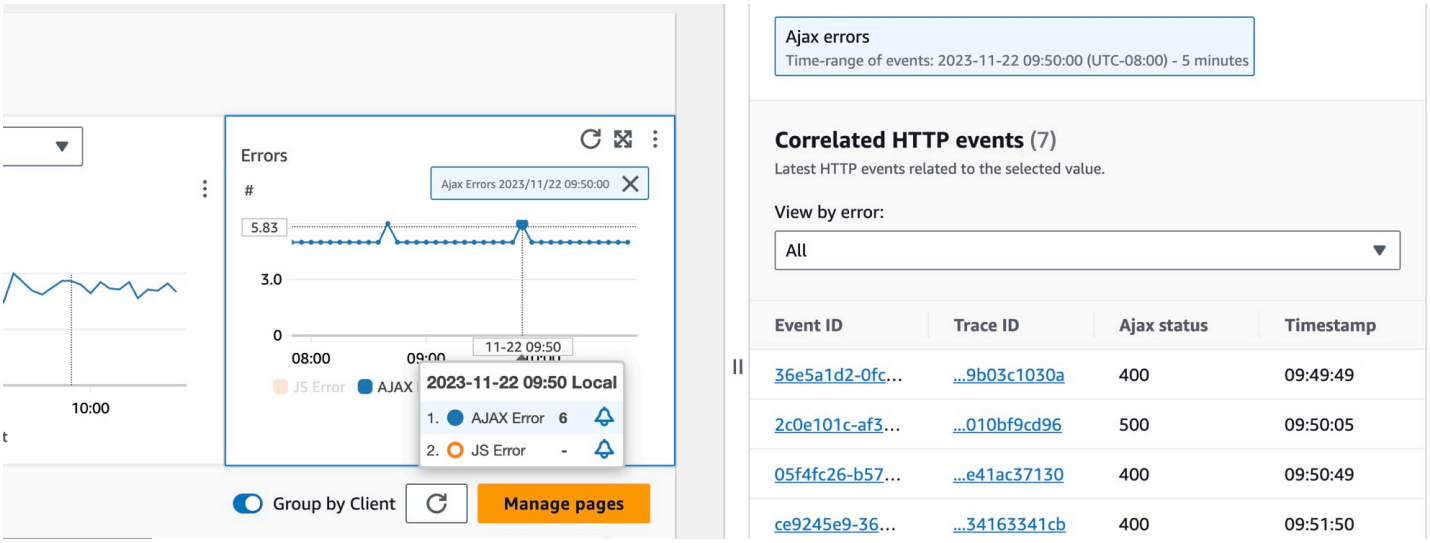
質を測定するときは、選択したクライアントページの、一連のメトリクスを使用します。これらのメトリクスには、ページロード、ウェブバイタル、エラーなどが含まれます。

テーブルにクライアントページを表示するには、CloudWatch RUM ウェブクライアントを X-Ray トレースに設定し、クライアントページの Application Signals メトリクスをオンにします。[ページの管理] をクリックして、Application Signals メトリクスを有効にするページを選択します。

フィルターテキストボックスを使用して、フィルターテキストボックスの下で、関心のあるクライアントページまたはアプリケーションモニターを特定します。[フィルターのクリア] を選択すると、テーブルのフィルターを削除できます。[クライアント別にグループ化] を選択すると、クライアントページをクライアントごとにグループ化できます。グループ化したら、クライアント名の横にある [+] アイコンを選択して行を展開し、そのクライアントのすべてのページを表示します。



クライアントページを選択するには、[クライアントページ] テーブルのクライアントページの横にあるオプションを選択します。詳細なメトリクスを表示する一連のグラフが表示されます。グラフ内のポイントにカーソルを合わせると、ポップアップが開いて詳しい情報が表示されます。グラフ内のポイントを選択すると、そのグラフ内の選択したポイントの相関パフォーマンスナビゲーションイベントを示す診断ペインが開きます。ナビゲーションイベントのリストからイベント ID を選択し、選択したイベントの CloudWatch RUM ページビューを開きます。



> **Note**
>
> クライアントページ内の AJAX エラーを確認するには、CloudWatch RUM ウェブクライアントバージョン 1.15 以降を使用してください。
> 現在、サービスごとに、最大 100 のオペレーション、Canary、クライアントページと、最大 250 の依存関係を表示できます。

CloudWatch Service Map を使用してアプリケーションのトポロジを表示し、運用状態をモニタリングする

> **Application Signals は Amazon CloudWatch のプレビューリリースであり、変更される可能性があります。**

> **Note**
>
> CloudWatch Service Map が ServiceLens map の代替になります。AWS X-Ray トレースに基づいてアプリケーションのマップを表示するには、X-Ray トレースマップを開きます。CloudWatch コンソールで、左側のナビゲーションペインから [X-Ray トレース] の下の [トレースマップ] を選択します。

Service Map を使用して、アプリケーションクライアント、Synthetics Canary、サービス、依存関係のトポロジを表示し、運用状態をモニタリングします。Service Map を表示するには、CloudWatch コンソールを開き、左側のナビゲーションペインの [Application Signals] セクションで [Service Map] を選択します。

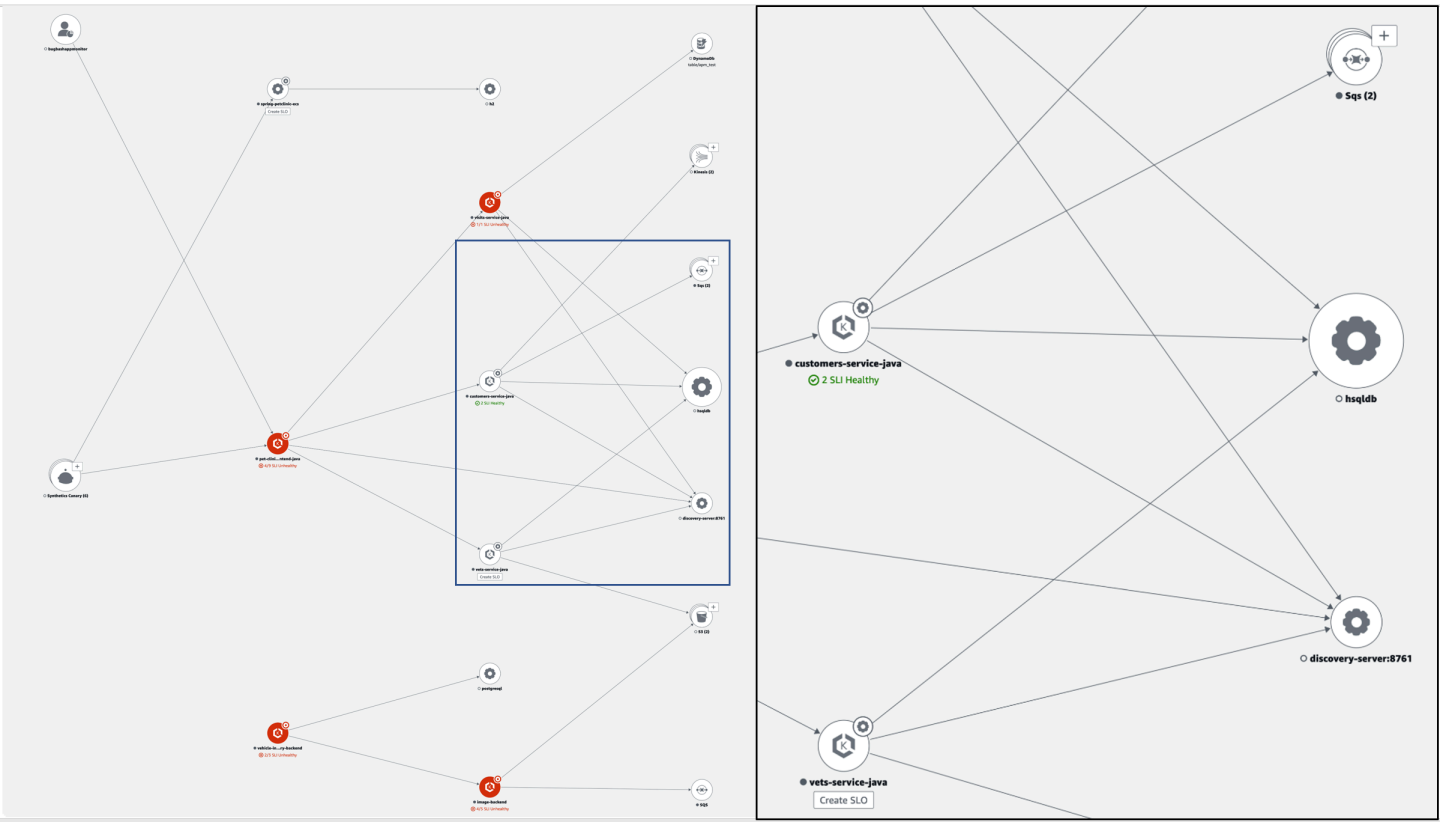
アプリケーションで Application Signals を有効にすると、Service Map を使用してアプリケーションの運用状態を簡単に監視できるようになります。

- クライアント、Canary、サービス、依存関係ノード間の接続を表示して、アプリケーションのトポロジと実行フローを把握しやすくします。これは、サービスオペレーターが開発チームではない場合に特に役立ちます。
- どのサービスがサービスレベル目標 (SLO) を達成しているのか、または達成していないのかを確認できます。サービスが SLO を達成していない場合は、ダウンストリームのサービスや依存関係

が問題の原因となっているのか、複数のアップストリームのサービスに影響を与えているのかをすばやく特定できます。

- 個々のクライアント、Synthetics Canary、サービス、または依存関係ノードを選択すると、関連するメトリクスが表示されます。[サービスの詳細] ページには、オペレーション、依存関係、Synthetics Canary、クライアントページに関する詳細が表示されます。
- Service Map をフィルタリングしてズームすると、アプリケーションのトポロジの一部に焦点を合わせたり、マップ全体を見たりしやすくなります。フィルターテキストボックスから 1 つ以上のプロパティを選択して、フィルターを作成します。各プロパティを選択すると、フィルター条件が表示されます。フィルターテキストボックスの下に、すべてのフィルターが表示されます。[フィルターのクリア] を選択すると、いつでもフィルターを削除できます。

次のサービスマップの例には、エッジを、インタラクションするコンポーネントに接続するサービスが示されています。SLO が定義されている場合、サービスマップにはヘルスステータスも表示されます。

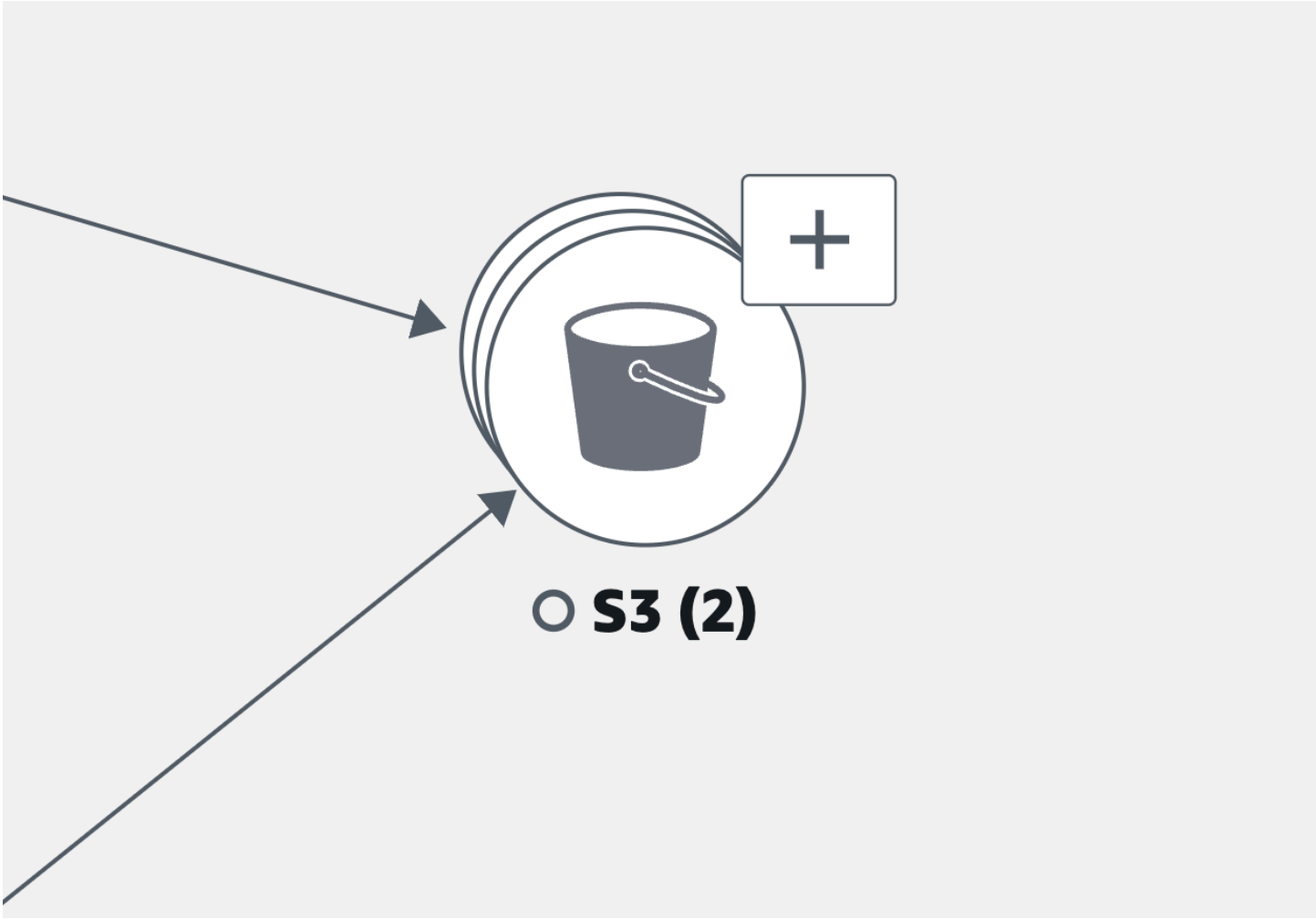


Service Map を詳しく知る

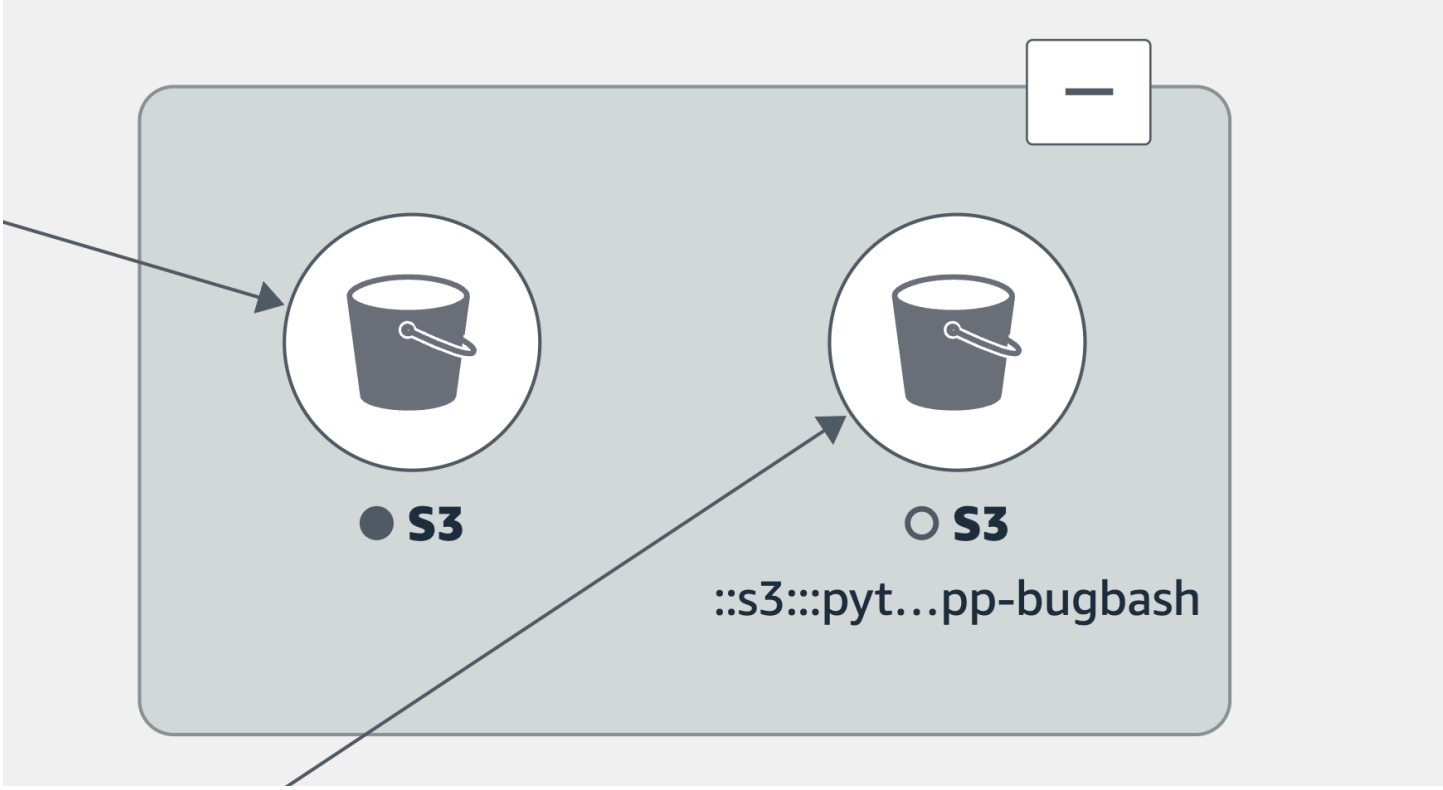
アプリケーションで Application Signals を有効にすると、Service Map にはサービスと依存関係を表すノードが表示されます。

CloudWatch RUM クライアントと Synthetics Canary のアクティブトレースを有効にすると、マップ上にクライアントノードと Canary ノードが表示されます。

デフォルトでは、Canary、RUM クライアント、同じ種類のAWS サービス依存関係は、サービスマップ内の、1 つの拡張可能アイコンにグループ化されます。AWS の外部のサービス依存関係は、デフォルトではグループ化されません。例えば、次の画像では、すべての Amazon S3 バケットが 1 つの拡張可能アイコンの下でグループ化されています。



前の画像では、Amazon S3 グループと発信元のサービスの間のラベルに、依存関係のアイコンの下の、括弧内のグループのエッジ数が示されています。(+) アイコンをクリックすると、次の図に示すように、グループが展開され、個々の要素が表示されます。

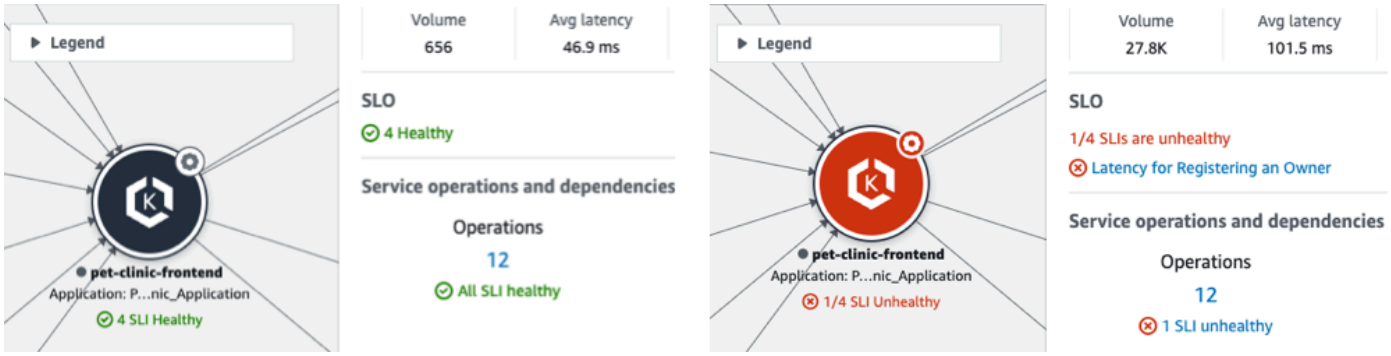


各ノードの種類とノード間のエッジ (接続) の詳細を見るには、次のタブをクリックします。

View your application services

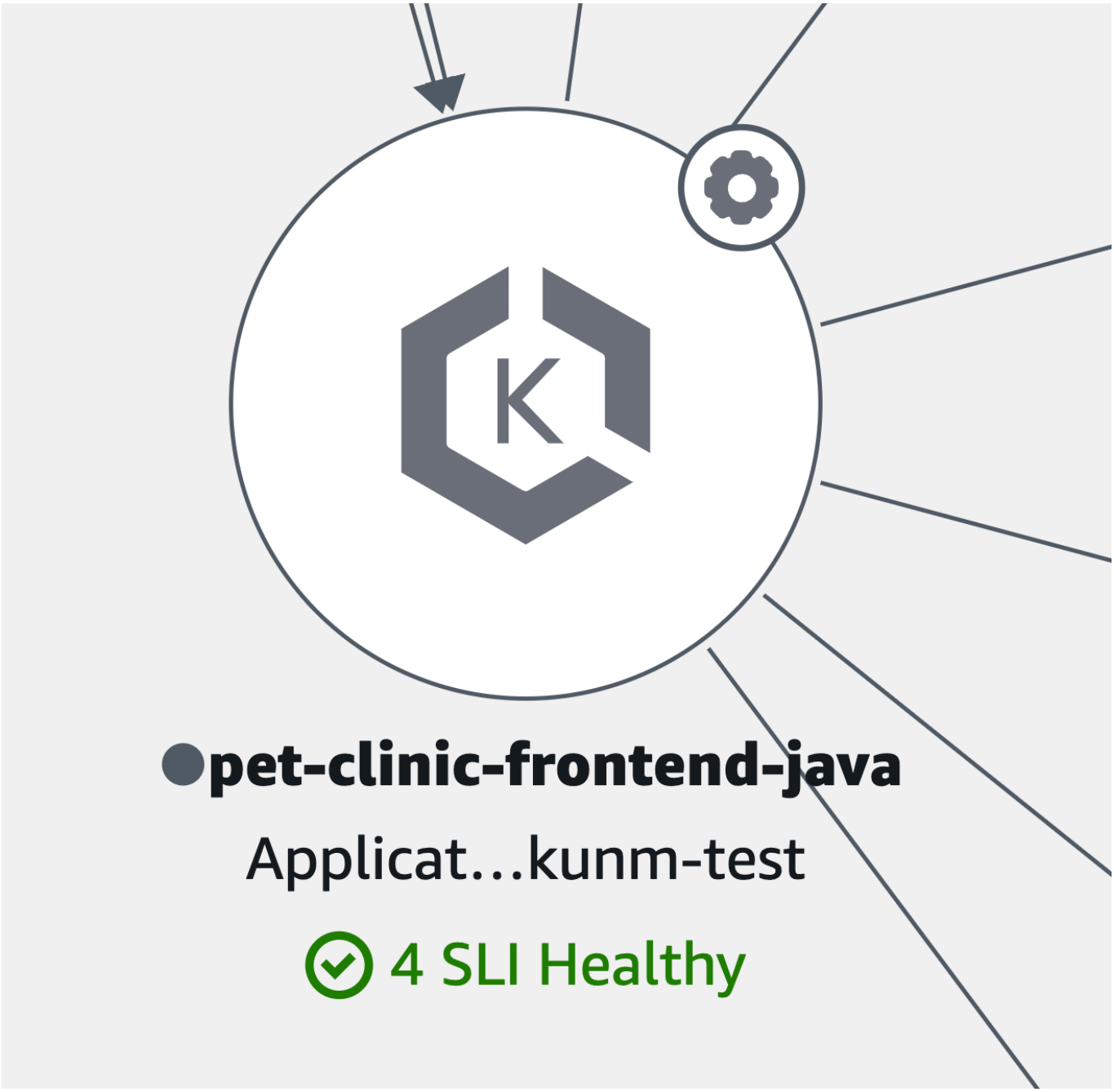
アプリケーションのサービスと、それらの SLO およびサービスレベル指標 (SLI) の状態は、Service Map で確認できます。サービスの SLO が作成されていない場合は、サービスノードの下にある [SLO の作成] ボタンをクリックします。

Service Map には、サービスがすべて表示されます。また、次の図に示すように、サービスを使用している顧客および Canary、ならびにそのサービスが呼び出す依存関係も表示されます。



次のアイコンは、Service Map 内のアプリケーションサービスの例を示しています。

- Amazon Elastic Kubernetes Service:



- Kubernetes コンテナ：

- Amazon Elastic Compute Cloud (Amazon EC2)

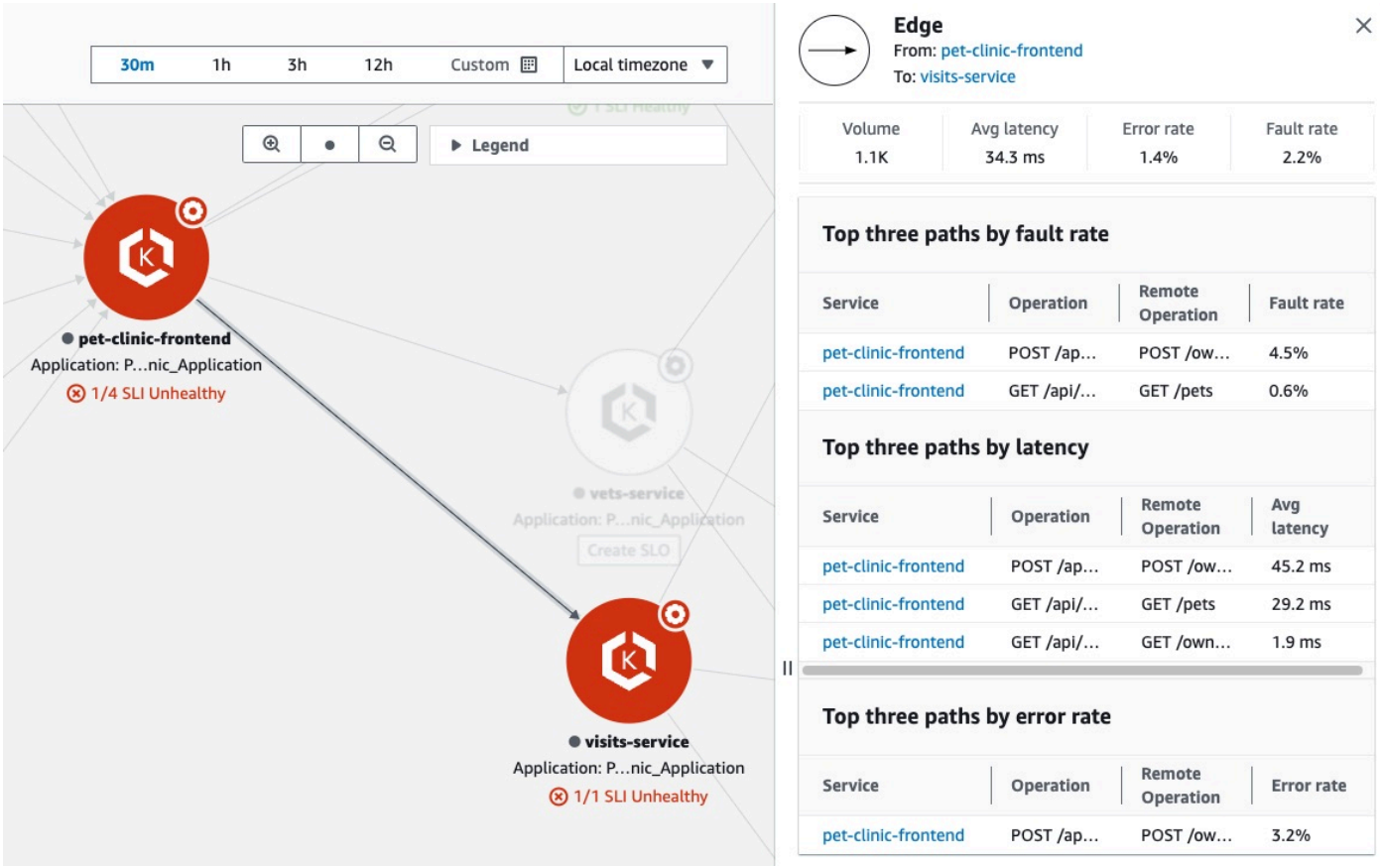
- これまではリストされていなかったその他のアプリケーションサービスタイプ：

サービスノードを選択すると、ペインが開き、次のような詳細なサービス情報が表示されます。

- 呼び出し量、レイテンシー、エラー、障害率のメトリクス。
- healthy または unhealthy である SLI と SLO の数。
- SLO に関する詳細情報を表示するオプション。
- サービスオペレーション、依存関係、Synthetics Canary、およびクライアントページの数。
- 各番号をクリックして [サービスの詳細] ページを開くオプション。
- AppRegistry または AWS Management Consoleホームページの Applications カードを使用して、基盤となるコンピューティングリソースをアプリケーションに関連付けた場合のアプリケーション名。

- アプリケーション名を選択すると、myApplications コンソールページにアプリケーションの詳細が表示されます。
- Amazon EKS でホストされているサービスの場合は `Cluster`、`Namespace`、`Workload`、Amazon ECS または Amazon EC2 でホストされているサービスの場合は `Environment` です。Amazon EKS がホストするサービスの場合は、任意のリンクを選択して CloudWatch Container Insights を開きます。

サービスノードとダウンストリームサービスまたは依存関係ノードの間の、エッジまたは接続を選択します。これにより、次の画像の例に示すように、上位パスを含むペインが障害率、レイテンシー、エラー率ごとに開きます。ペイン内の任意のリンクを選択すると、[サービスの詳細] ページが開き、選択したサービスまたは依存関係の詳細情報が表示されます。

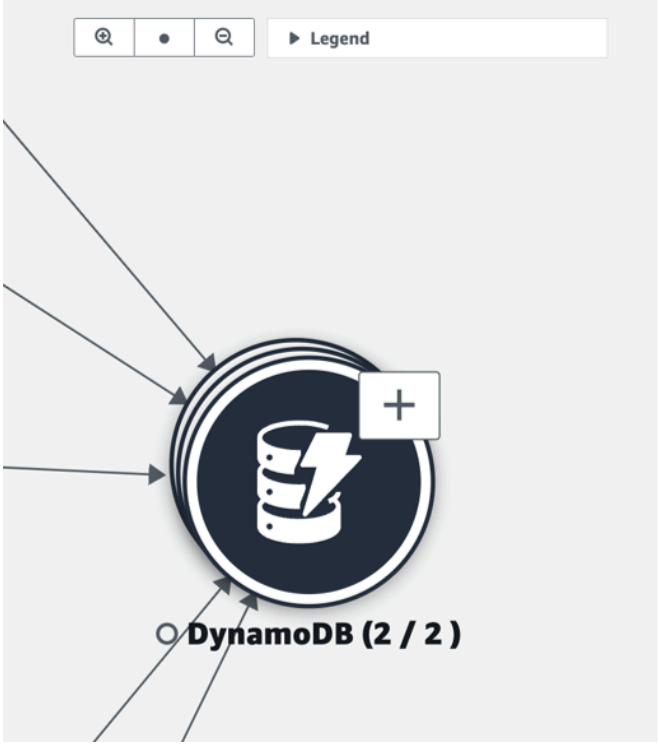


View dependencies

アプリケーションの依存関係は、それを呼び出すサービスに接続された Service Map に表示されます。

依存関係ノードを選択すると、障害率、レイテンシー、エラー率別に上位パスが表示されたペインが開きます。サービスまたはターゲットのリンクをクリックすると [サービスの詳細] ページが

開き、以下の画像の例に示すように、選択したサービスまたは依存関係のターゲットに関する詳細情報が表示されます。



Volume	Avg latency	Error rate	Fault rate
-	-	-	-

Top three paths by fault rate

Service	Remote operation	Fault rate
No paths with faults		

Top three paths by latency

Service	Remote operation	Avg latency
billing-service-ec2-python	PutItem	282.8 ms
billing-service-python	PutItem	75.6 ms
visits-service-java	PutItem	64.9 ms

Top three paths by error rate

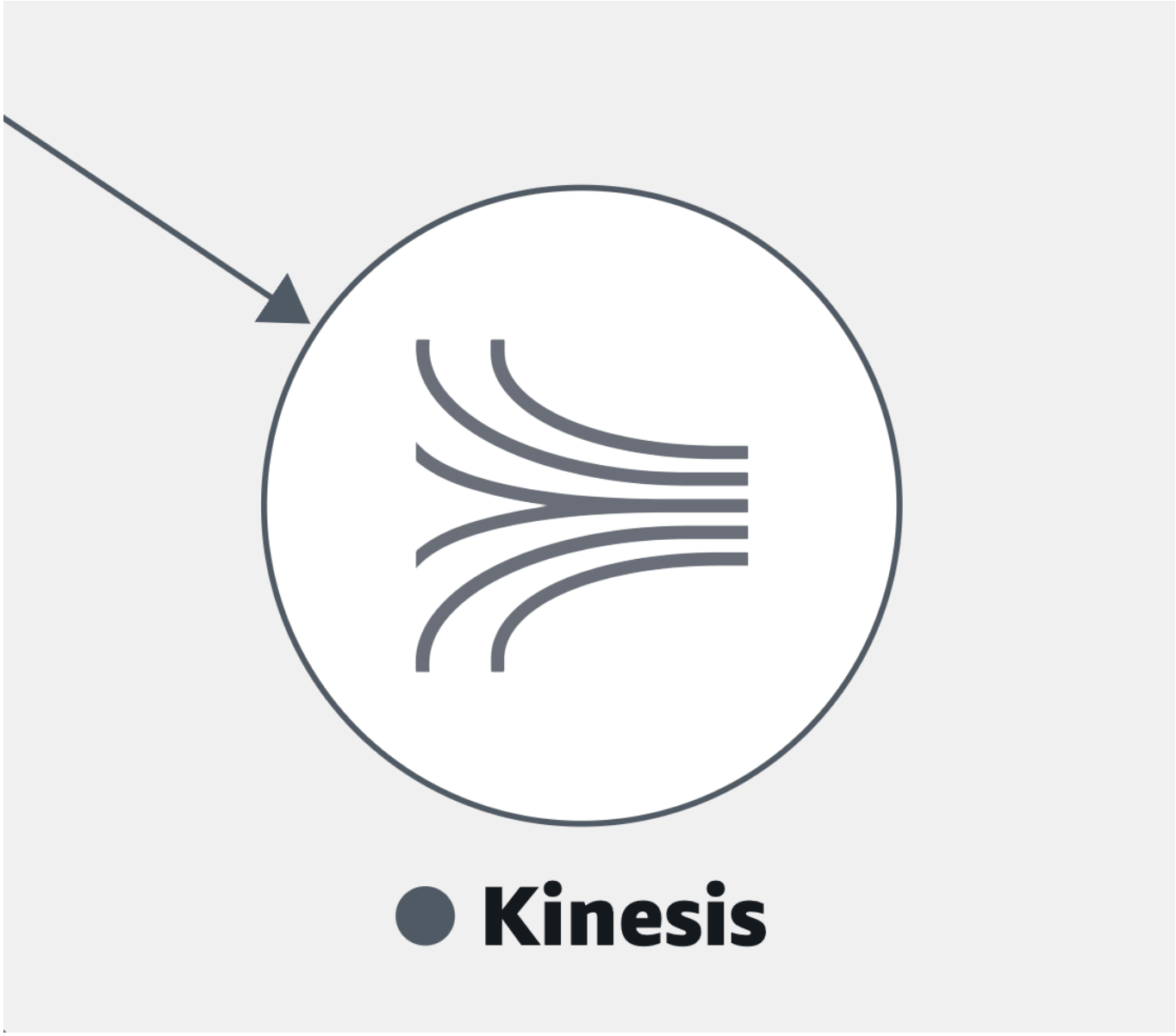
Service	Remote operation	Error rate
visits-service-java	PutItem	9.6%

サービスの依存関係は、デフォルトで 1 つの拡張可能なアイコンにグループ化されます。(+) アイコンをクリックすると、次の図に示すように、グループが展開して個々の要素が表示されます。

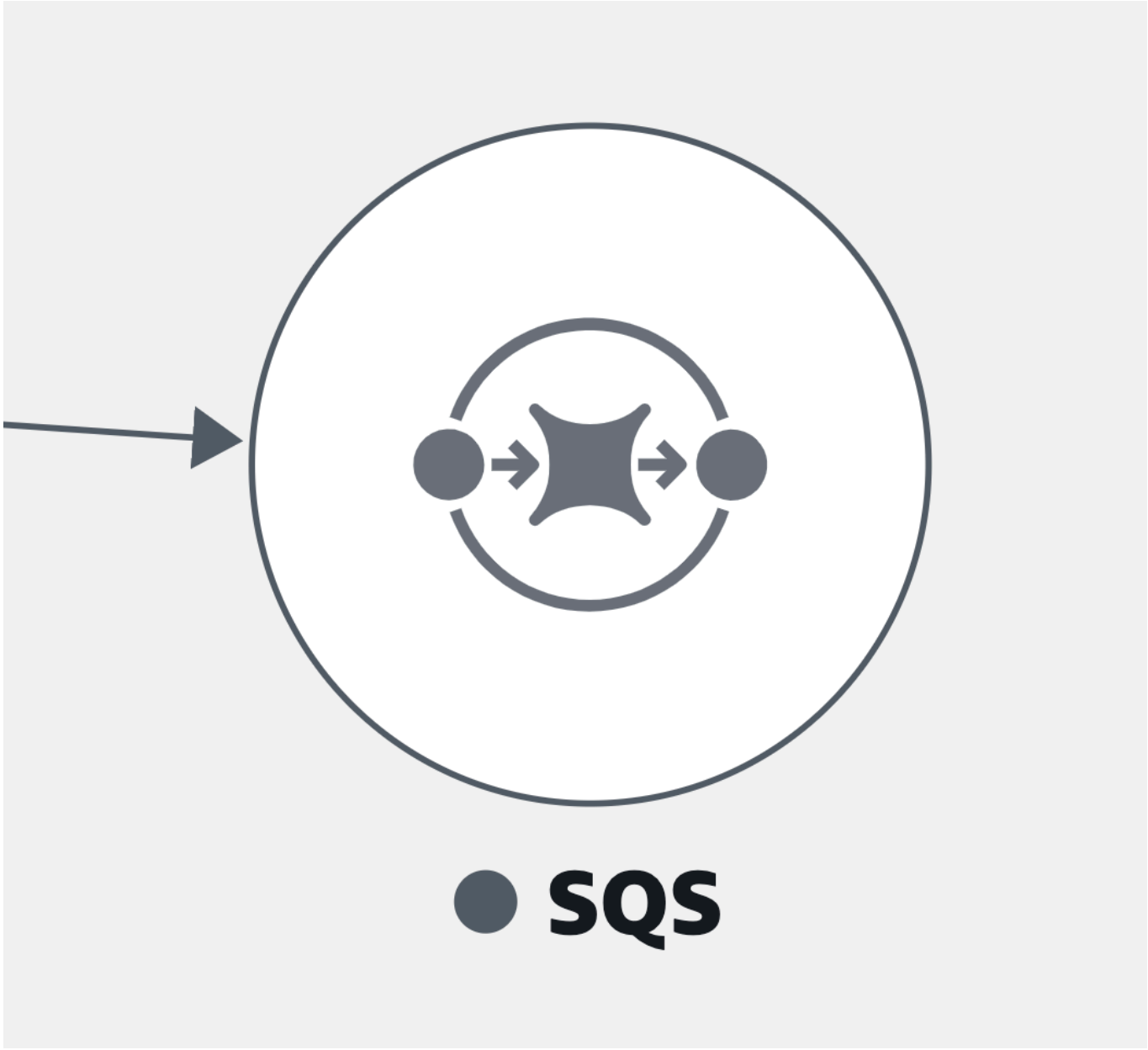
次のアイコンは、サービスマップ内の依存関係ノードの例を示しています。

- Amazon S3 バケット

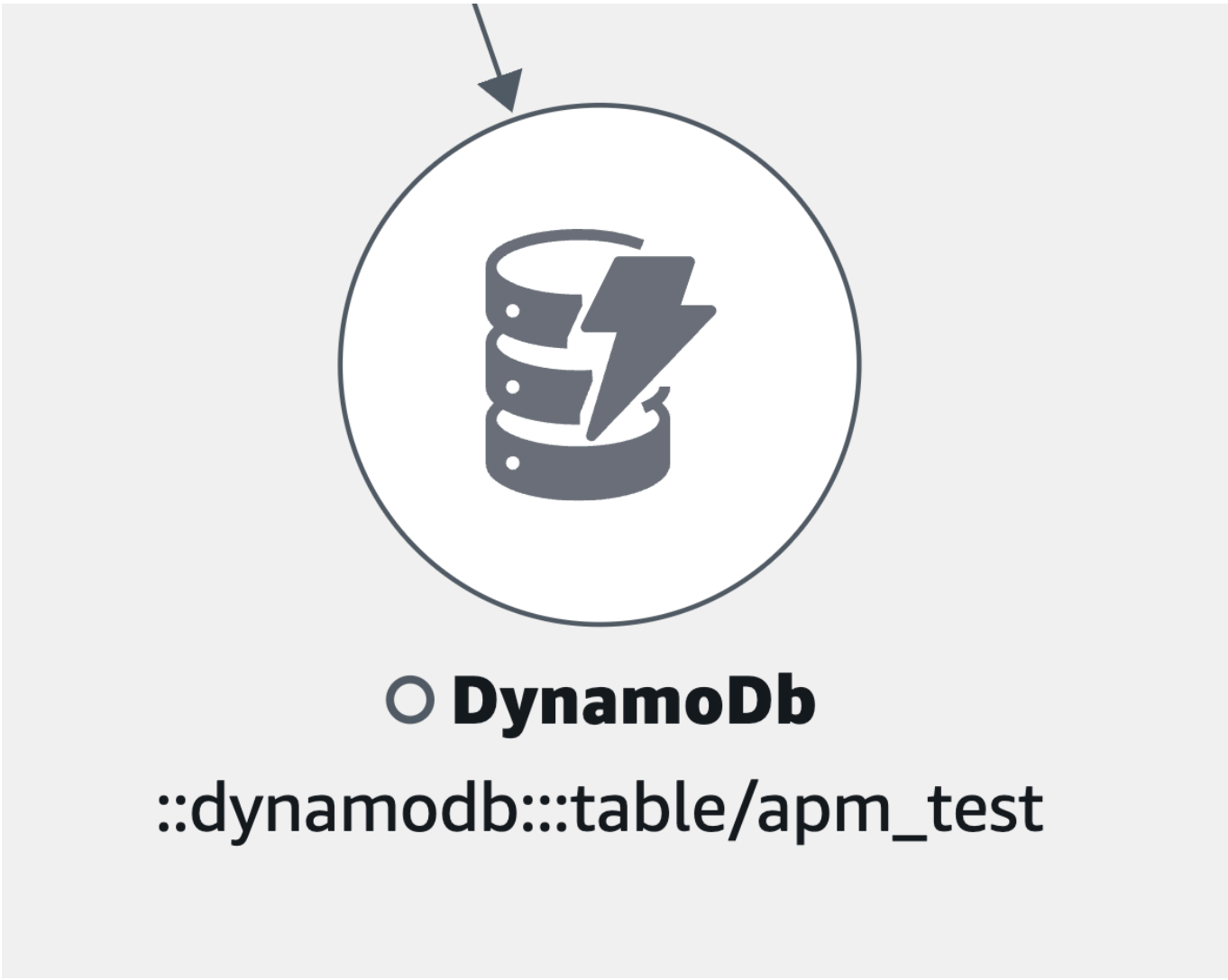
- Amazon Kinesis ストリーム:



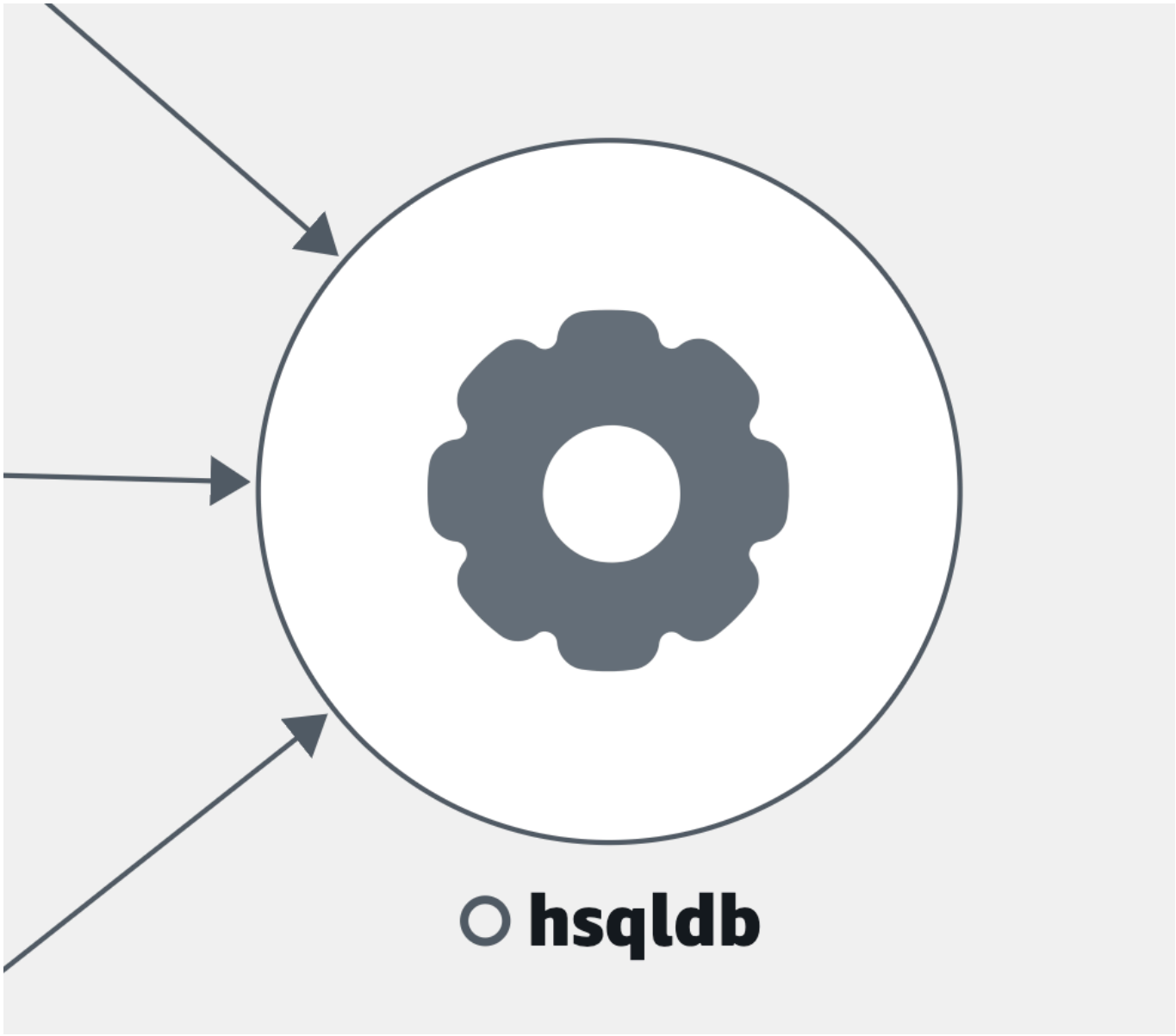
- Amazon Simple Queue Service (Amazon SQS)



- Amazon DynamoDB テーブル:



- これまでリストされたことがないその他の依存関係タイプ：



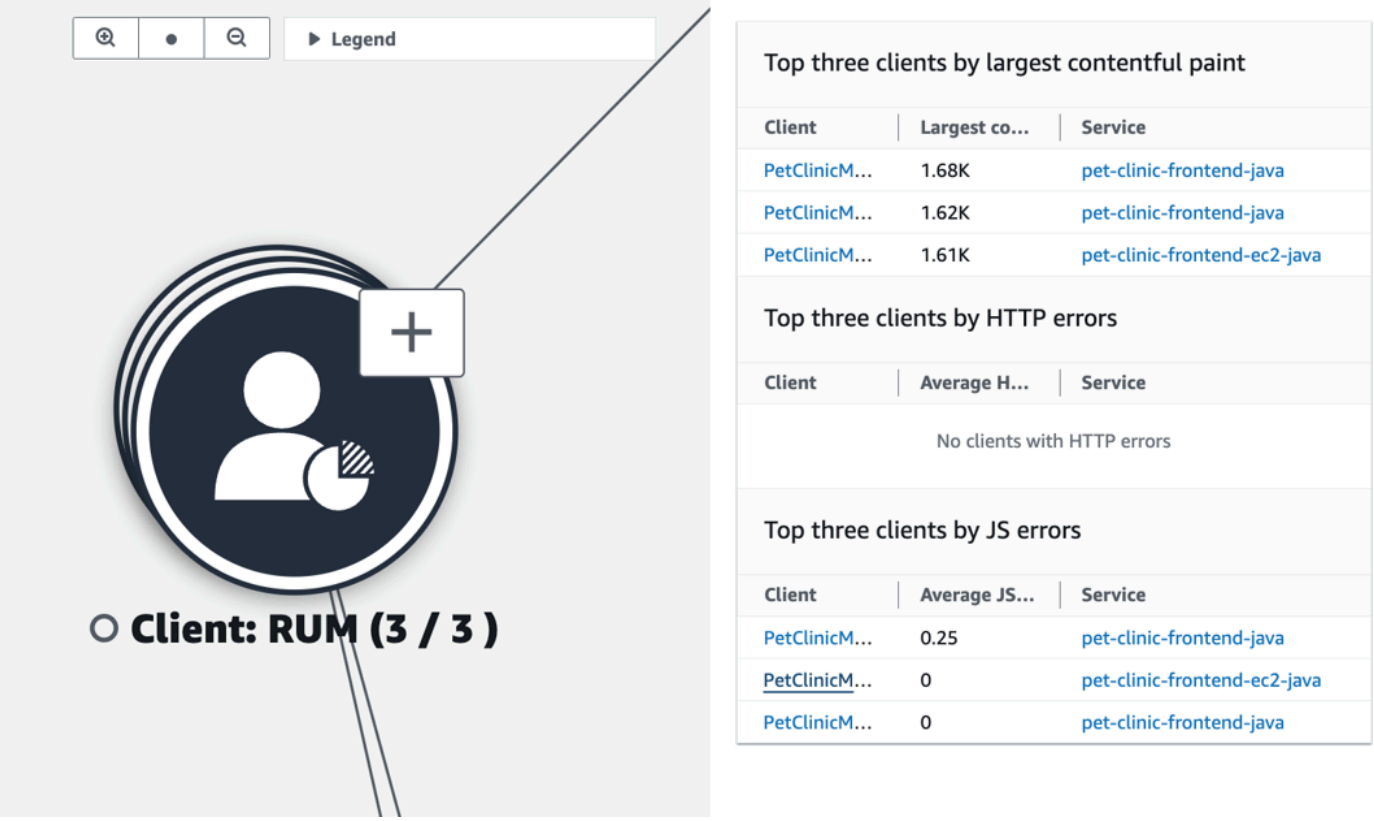
View clients

CloudWatch RUM ウェブクライアントの X-Ray トレースを有効にすると、ウェブクライアントが呼び出すサービスに接続された Service Map にクライアントが表示されます。

クライアントノードを選択すると、次のような詳細なクライアント情報が表示されているペインが開きます。

- ページロード、平均ロード時間、エラー、平均ウェブバイタルのメトリクス。
- エラーの内訳を示すグラフ。
- CloudWatch RUM でクライアントの詳細を表示するためのリンク。

RUM クライアントは、デフォルトで 1 つの拡張可能なアイコンにグループ化されます。(+) アイコンをクリックすると、次の図に示すように、グループが展開され、個々の要素が表示されます。



次のアイコンは、サービスマップ内の RUM クライアントの例を示しています。

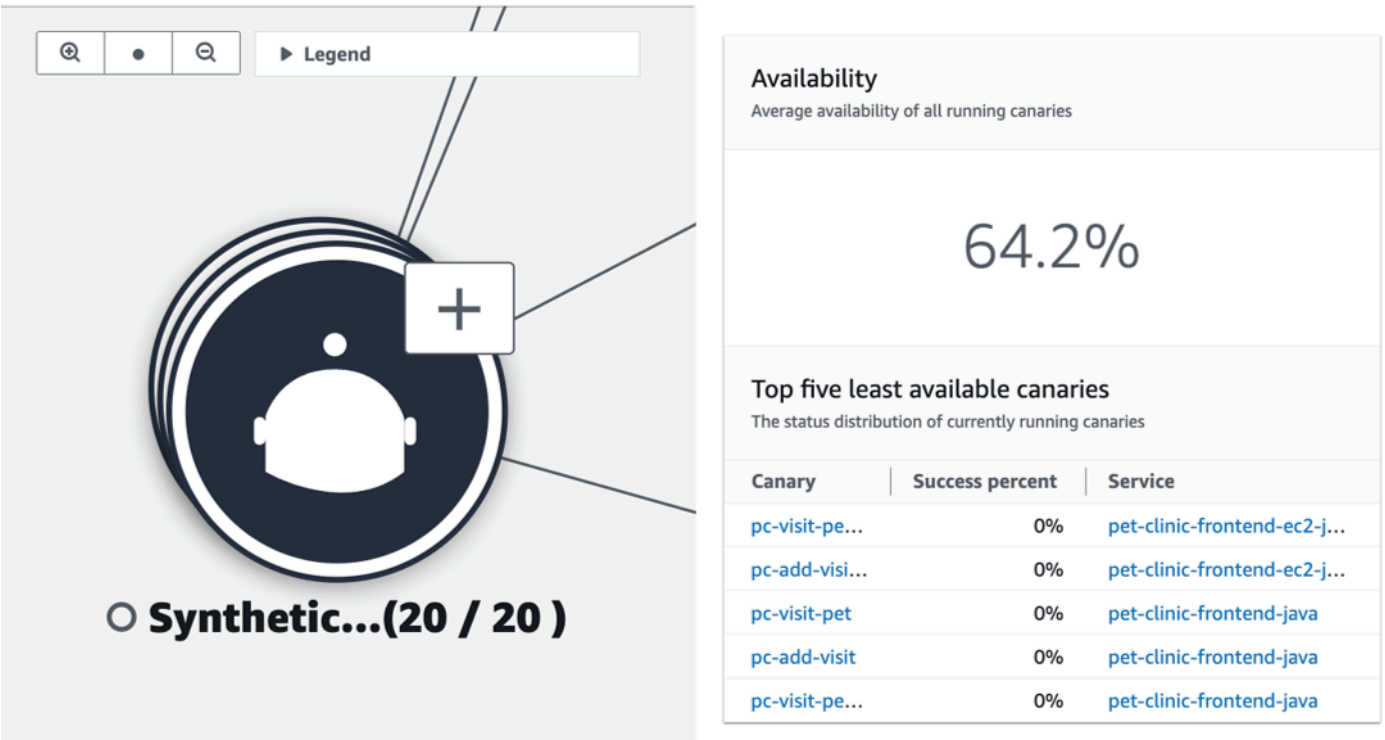
- RUM クライアント –

> **Note**
>
> クライアントページ内の AJAX エラーを確認するには、CloudWatch RUM ウェブクライアントバージョン 1.15 以降を使用してください。

View synthetics canaries

CloudWatch Synthetics Canary の AWS X-Ray トレースを有効にすると、以下に示すように、呼び出すサービスに接続された Service Map に Synthetics Canary が表示されます。

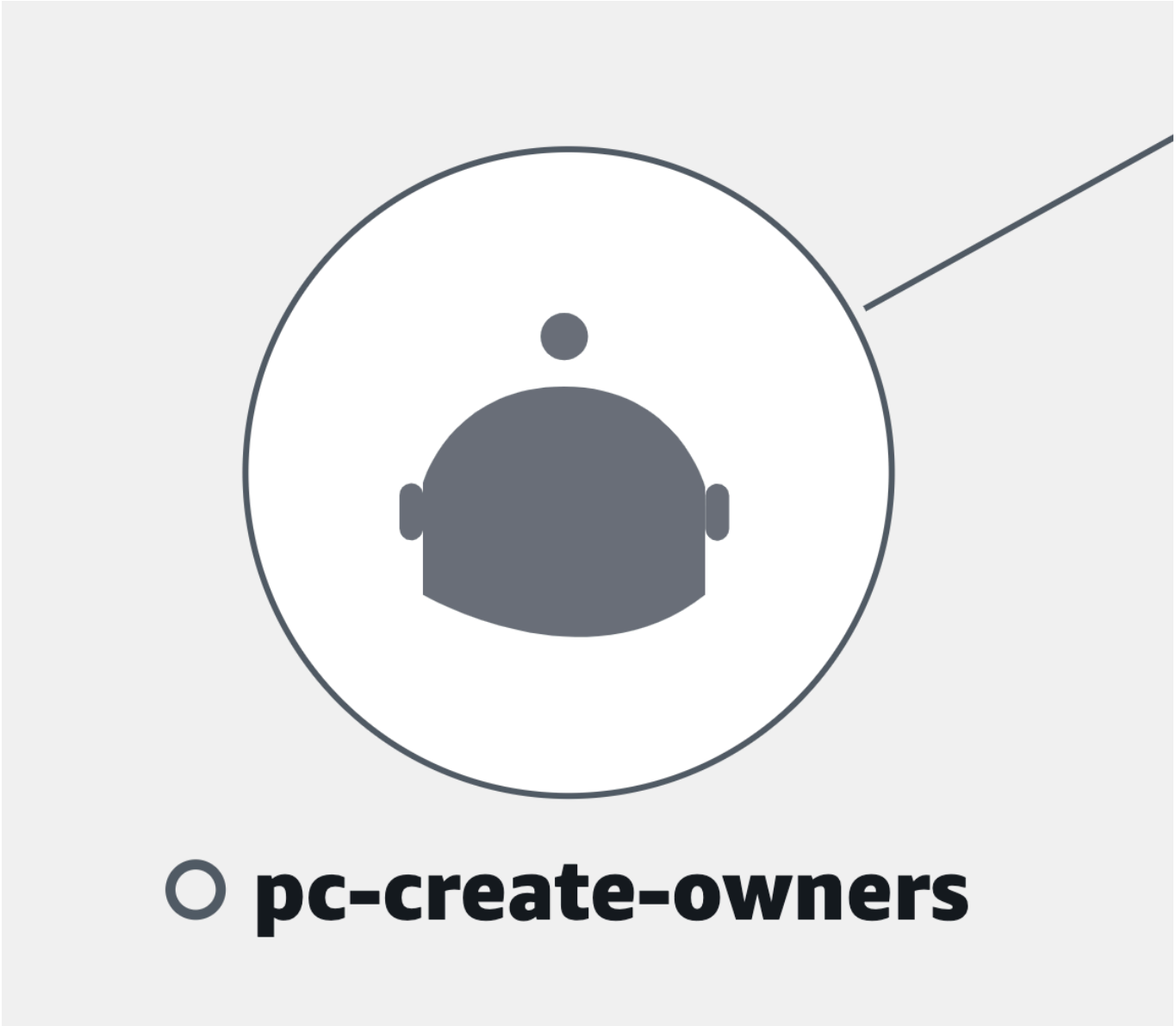
次の画像に示すように、Canary ノードをクリックして詳細な Canary 情報を表示するペインを開きます。

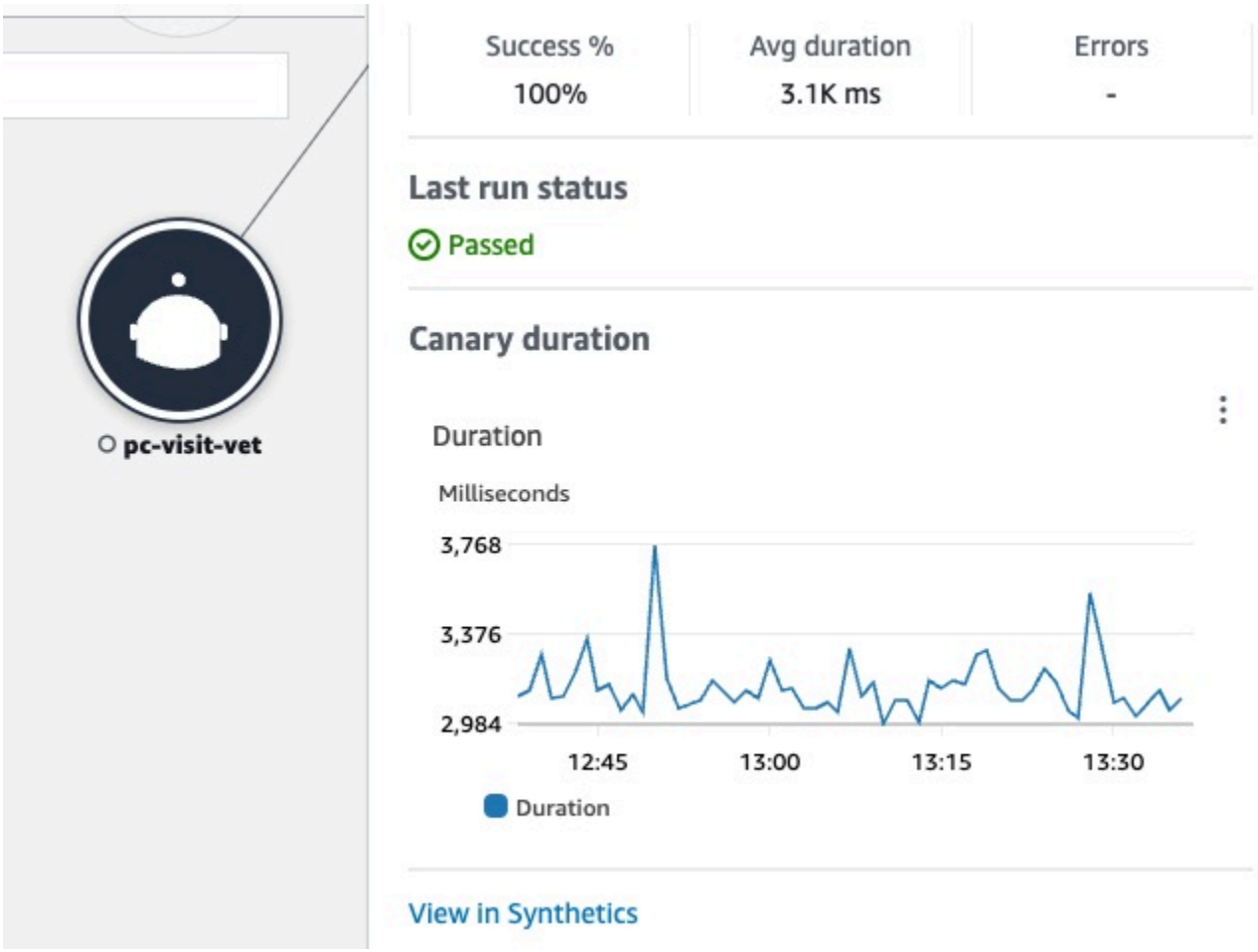


Canary は、デフォルトで 1 つの拡張可能なアイコンにグループ化されます。(+) アイコンをクリックすると、次の図に示すように、グループが展開して個々の要素が表示されます。

次のアイコンは、サービスマップ内のクライアントの例を示しています。

- Synthetics Canary -


pc-create-owners



Canary ノードのペインには、次の内容が表示されます。

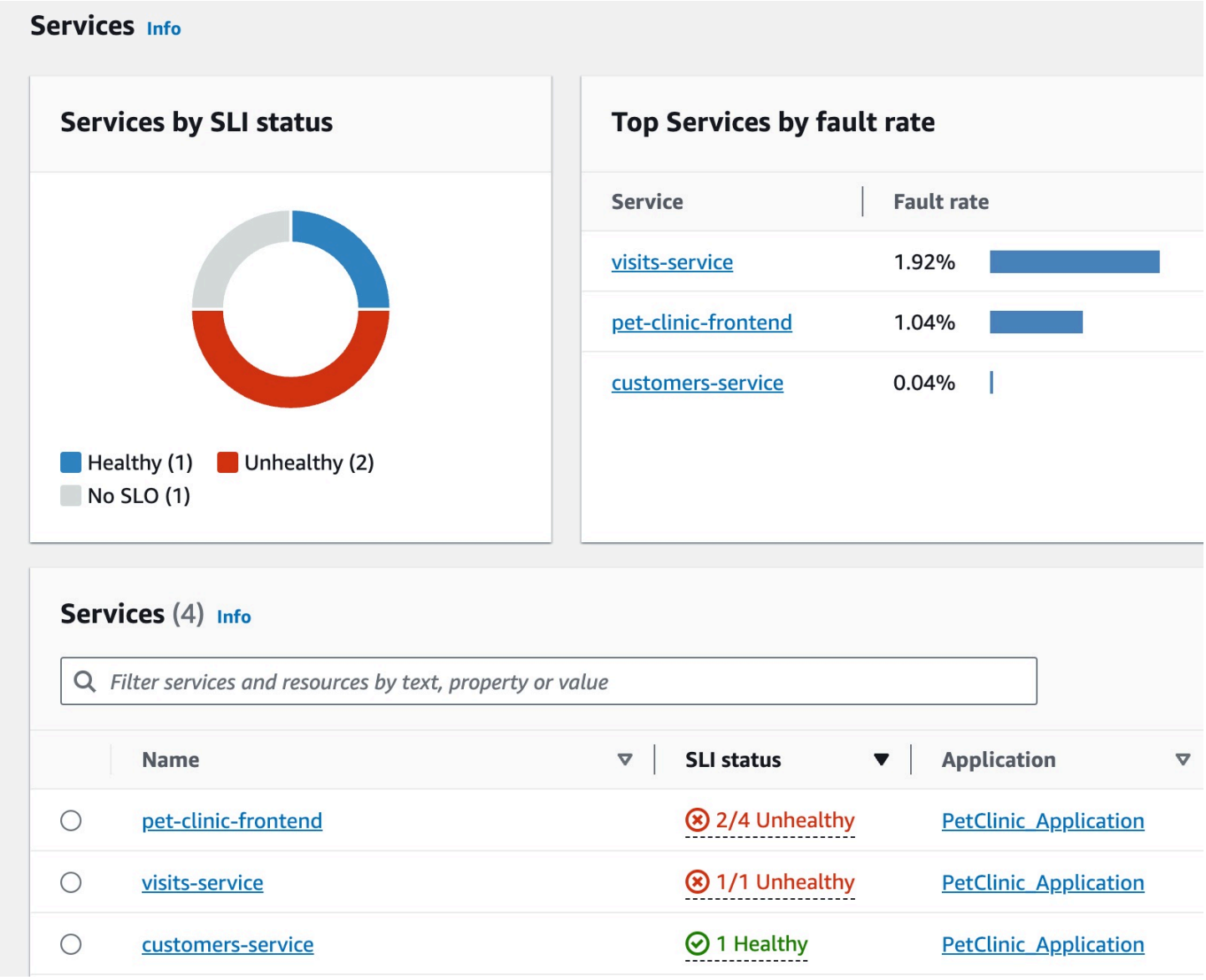
- 成功率、平均所要時間、エラーのメトリクス。
- 最終の Canary 実行のステータス。
- Canary 実行時間を表示しているグラフ。グラフシリーズにカーソルを合わせると、詳細情報を含むポップアップが表示されます。
- CloudWatch Synthetics で Canary の詳細を表示するためのリンク。

例: Application Signals を使用して、運用状態の問題を解決する

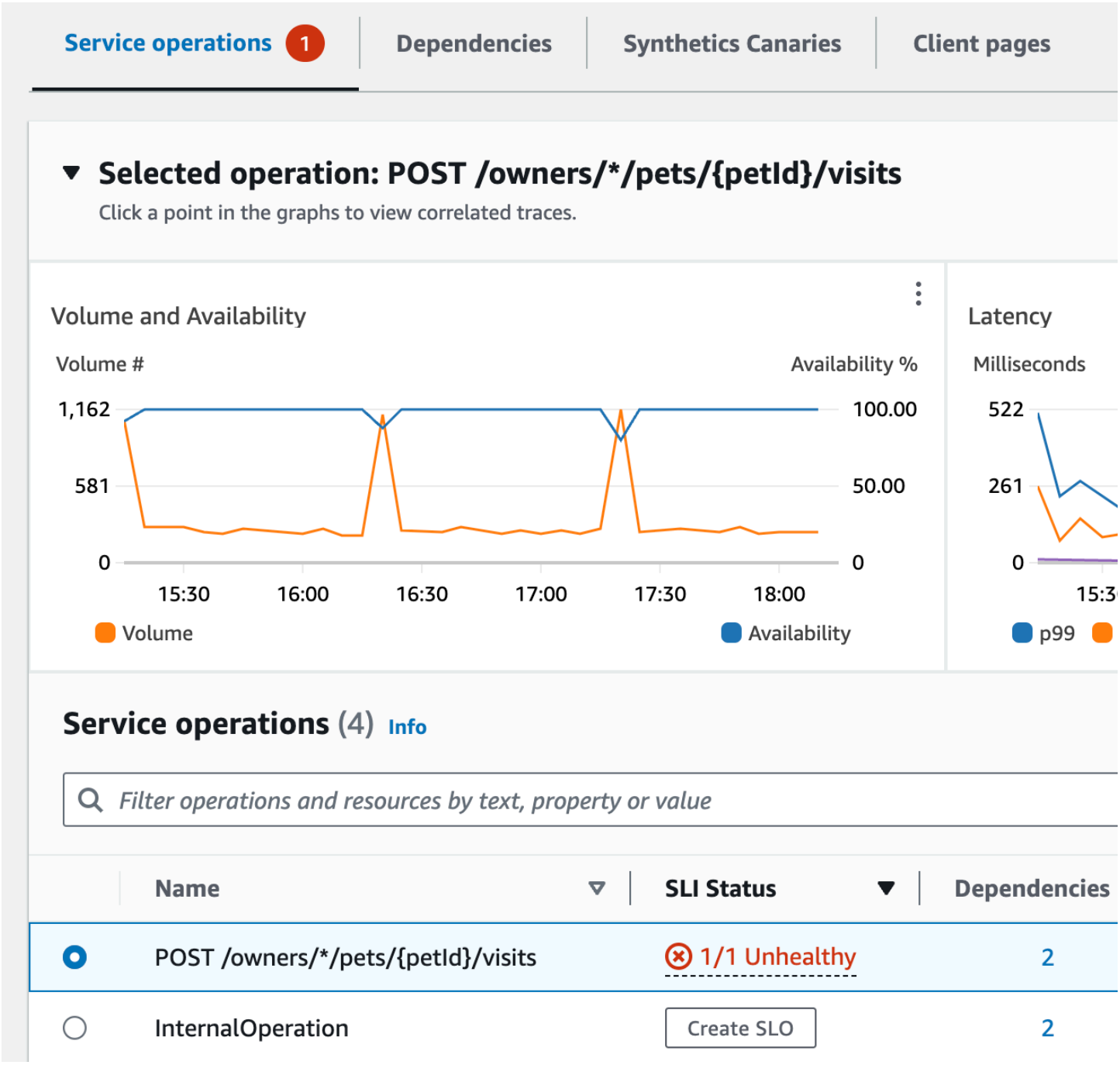
⚠ Application Signals は Amazon CloudWatch のプレビューリリースであり、変更される可能性があります。

以下のシナリオは、Application Signals を使用して、サービスをモニタリングし、サービス品質の問題を特定する方法の例を示しています。ドリルダウンして潜在的な根本原因を特定し、問題を解決するための対策を講じます。この例は、DynamoDB といった AWS のサービスを呼び出す複数のマイクロサービスで構成されるペットクリニックアプリケーションに焦点を当てています。

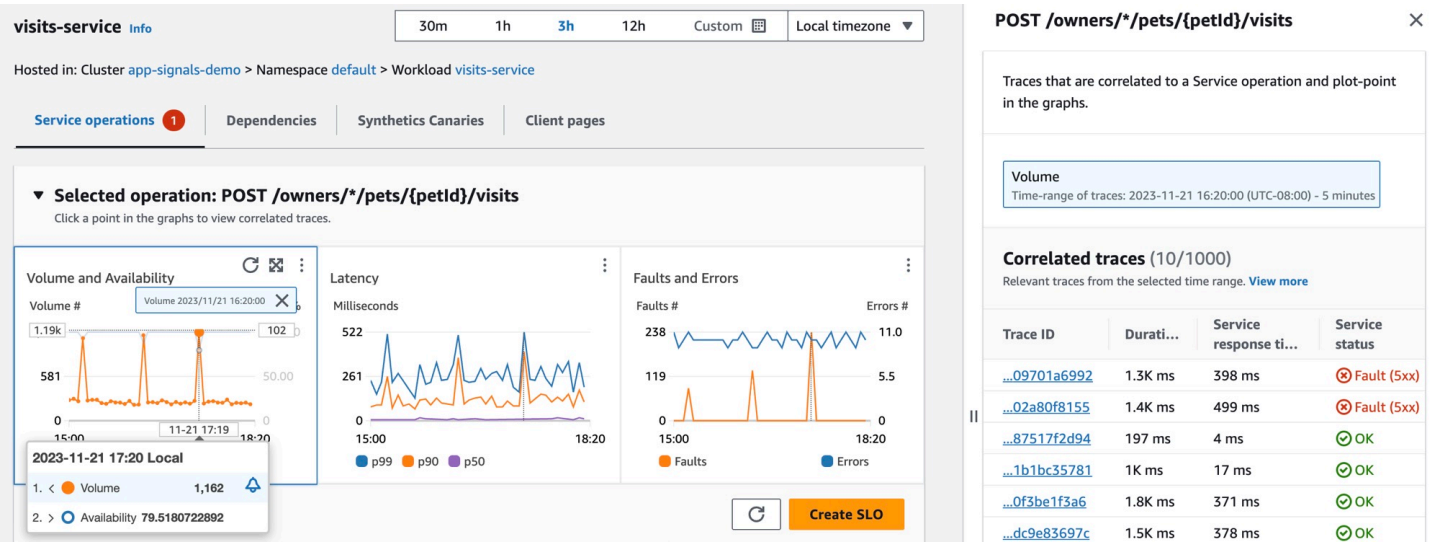
Jane は、ペットクリニックアプリケーションの運用状態を監督する DevOps チームの一員です。Jane のチームは、アプリケーションの可用性と応答性が高いことを確認することに全力を注いでいます。チームは、サービスレベル目標 (SLO) を使用して、これらのビジネス上のコミットメントに対するアプリケーションのパフォーマンスを測定します。彼女は、複数のサービスレベル指標 (SLI) が異常であるという警告を受け取ります。CloudWatch コンソールを開いてサービスページに移動すると、いくつかのサービスが異常な状態であることがわかりました。



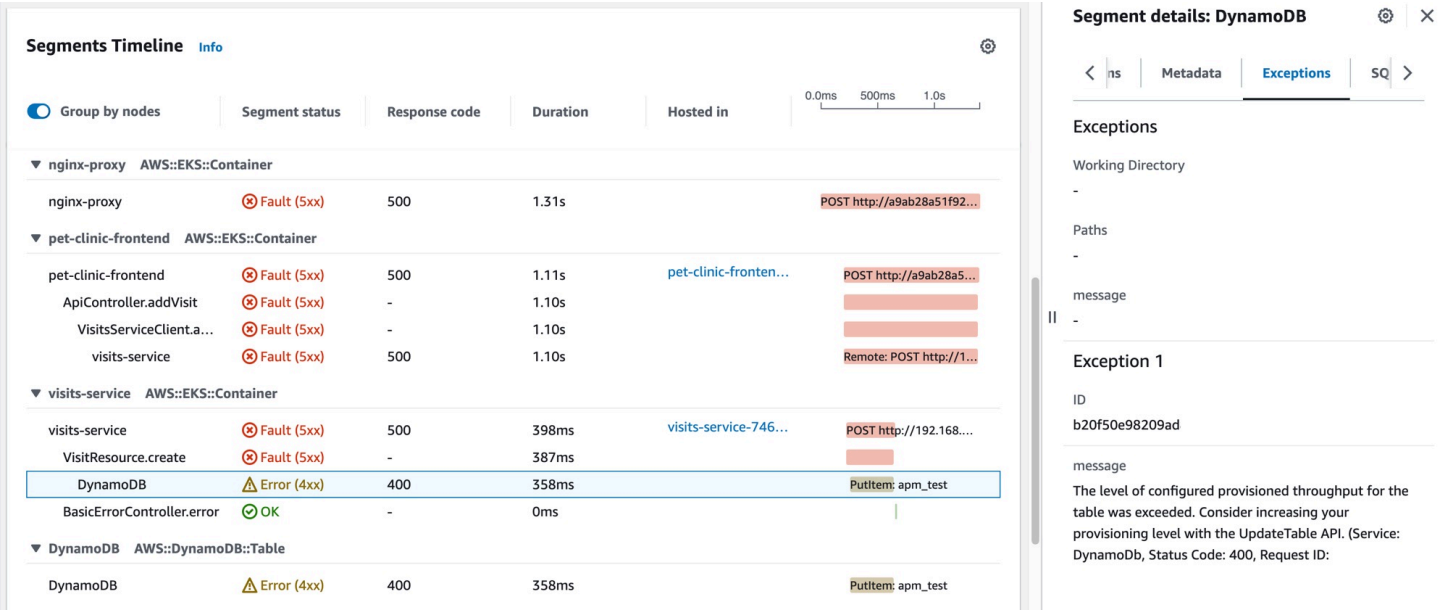
Jane は、ページの上部を見て、障害率で最上位のサービスが `visits-service` であることを知りました。彼女がグラフ内のリンクを選択すると、そのサービスのサービス詳細ページが開きました。サービスのオペレーションテーブルには、異常なオペレーションがありました。このオペレーションを選択すると、[量と可用性] グラフが表示されました。そのグラフから、定期的な呼び出し量の急増と可用性の一時的低下には相関関係があるらしいことがわかりました。



Jane は、サービスの可用性の一時的低下を詳しく調べるために、グラフ内の可用性データポイントの 1 つを選択しました。ドロワーが開き、選択したデータポイントと相関関係がある X-Ray トレースが表示されました。彼女は、障害を含むトレースが複数あることに気付きました。



Jane が障害ステータスと相関関係があるトレースの 1 つを選択すると、選択したトレースの X-Ray トレース詳細ページが開きました。Jane は、[セグメントのタイムライン] セクションまでスクロールし、DynamoDB テーブルへの呼び出しがエラーを返していることがわかるまで呼び出しパスをたどりました。DynamoDB セグメントを選択し、右側のドロワーの [例外] タブに移動しました。



Jane は、DynamoDB リソースの設定に誤りがあるため、顧客のリクエストが急増するとエラーが発生することに気付きました。DynamoDB テーブルのプロビジョニングされたスループットのレベルが定期的に超過していたため、サービスの可用性に問題が生じ、SLI に異常が発生していたのです。

この情報に基づいて、彼女のチームはプロビジョニングされたスループットをより高いレベルに設定し、アプリケーションの高可用性を確保できるようになりました。

収集される標準アプリケーションメトリクス

⚠ Application Signals はプレビューリリース中です。この機能についてご意見がありましたら、app-signals-feedback@amazon.com までご連絡ください。

Application Signals では、検出したサービスから標準アプリケーションメトリクスを収集します。こうしたメトリクスからは、サービスのパフォーマンスに最も影響を与える要因、例えば、レイテンシー、障害、エラーに関する情報を得られます。これにより、問題の特定、パフォーマンス傾向のモニタリング、リソースの最適化を行い、全体的なユーザーエクスペリエンスを向上させることができます。

次の表に、Application Signals によって収集されるメトリクスを示します。こうしたメトリクスは AppSignals 名前空間の CloudWatch に送信されます。

メトリクス	説明
Latency	リクエストの発生からデータ転送が開始されるまでの時間的な差。 単位: ミリ秒
Faults	サーバー側の障害 (HTTP 5XX) と、OpenTelemetry のスパンステータスエラーを合わせた件数。 単位: なし
Errors	クライアントによるエラー (HTTP 4XX) の件数。これらは、サービス側での問題ではなく、リクエスト上のエラーとされるため、Application Signals ダッシュボードに表示される Availability メトリクスでは、こうしたエラーはサービス障害とは見なされません。 単位: なし

Application Signals のダッシュボードに表示される Availability メトリクスは、(1 - **Faults**/Total)*100 という計算式によって求めます。Successful の応答数とは、5XX エラーのない応答の総件数を意味し、4XX 応答は、Availability の計算時に Successful の応答として処理されます。

収集されるディメンションと、ディメンションの組み合わせ

各標準アプリケーションメトリクスには、次のディメンションが定義されています。ディメンションの詳細については、「ディメンション」を参照してください。

収集されるディメンションは、サービスメトリクスと依存関係メトリクスごとに異なります。例えば、Application Signals によって検出されたサービス内で、マイクロサービス A がマイクロサービス B を呼び出し、マイクロサービス B がそのリクエストを処理するとします。この場合、マイクロサービス A が依存関係メトリクスを出力し、マイクロサービス B がサービスメトリクスを出力します。クライアントがマイクロサービス A を呼び出す場合は、マイクロサービス A がそのリクエストを処理し、サービスメトリクスを生成します。

サービスメトリクスのディメンション

次のディメンションがサービスメトリクスとして収集されます。

ディメンション	説明
Service	サービス名
Operation	API オペレーションなどの操作名
HostedIn.EKS.Cluster	サービスが稼働している Amazon EKS クラスターの名前 このディメンションは、サービスが Amazon EKS で稼働している場合にのみ収集される
HostedIn.K8s.Namespace	サービスが稼働している Kubernetes 名前空間の名前 このディメンションは、サービスが Amazon EKS で稼働している場合にのみ収集される
HostedIn.Environment	ユーザーが定義した、サービスの稼働環境名 このディメンションは、Amazon EKS ではない環境でサービスが稼働している場合にのみ収集される

CloudWatch コンソールで、これらのメトリクスを表示する場合、次のようなディメンションの組み合わせで表示することもできます。

- `Service`, `Operation`, `HostedIn.EKS.Cluster`, `HostedIn.K8s.Namespace`
- `Service`, `HostedIn.EKS.Cluster`, `HostedIn.K8s.Namespace`

Amazon EKS 以外のプラットフォームでは、次のようなディメンションの組み合わせでサービスメトリクスを表示することもできます。

- `Service`, `Operation`, `HostedIn.Environment`
- `Service`, `HostedIn.Environment`

依存関係メトリクスのディメンション

次のディメンションが依存関係メトリクスとして収集されます。

ディメンション	説明
`Service`	サービス名
`Operation`	API オペレーションなどの操作名
`RemoteService`	呼び出されるリモートサービスの名前
`RemoteOperation`	呼び出される API オペレーションの名前
`HostedIn.EKS.Cluster`	サービスが稼働している Amazon EKS クラスターの名前 このディメンションは、サービスが Amazon EKS で稼働している場合にのみ収集される
`HostedIn.K8s.Namespace`	サービスが稼働している Kubernetes 名前空間の名前 このディメンションは、サービスが Amazon EKS で稼働している場合にのみ収集される
`K8s.RemoteNamespace`	依存関係サービスが稼働している Kubernetes 名前空間の名前

ディメンション	説明
	このディメンションは、サービスが Amazon EKS で稼働している場合にのみ収集される
RemoteTarget	リモート呼び出しによって呼び出されるリソースの名前 リモート呼び出しが特定のリソースを対象としていない場合、このディメンションには値がありません。 このディメンションは、サービスが Amazon EKS で稼働している場合にのみ収集される
HostedIn.Environment	ユーザーが定義した、サービスの稼働環境名 このディメンションは、Amazon EKS ではない環境でサービスが稼働している場合にのみ収集される

CloudWatch コンソールで、これらのメトリクスを表示する場合、次のようなディメンションの組み合わせで表示することもできます。

いずれのプラットフォームでも実行可能

- RemoteService

Amazon EKS クラスターで実行可能

- Service, Operation, HostedIn.EKS.Cluster, HostedIn.K8s.Namespace, RemoteService, RemoteOperation, K8s.RemoteNamespace, RemoteTarget
- Service, Operation, HostedIn.EKS.Cluster, HostedIn.K8s.Namespace, RemoteService, RemoteOperation, K8s.RemoteNamespace
- Service, Operation, HostedIn.EKS.Cluster, HostedIn.K8s.Namespace, RemoteService, RemoteOperation, RemoteTarget
- Service, Operation, HostedIn.EKS.Cluster, RemoteService, RemoteOperation,
- Service, HostedIn.EKS.Cluster, HostedIn.K8s.Namespace, RemoteService, K8s.RemoteNamespace
- Service, HostedIn.EKS.Cluster, RemoteService, K8s.RemoteNamespace

- Service, HostedIn.EKS.Cluster, HostedIn.K8s.Namespace, RemoteService, RemoteOperation, K8s.RemoteNamespace, RemoteTarget
- Service, HostedIn.EKS.Cluster, HostedIn.K8s.Namespace, RemoteService, RemoteOperation, K8s.RemoteNamespace
- Service, HostedIn.EKS.Cluster, HostedIn.K8s.Namespace, RemoteService, RemoteOperation, RemoteTarget
- Service, HostedIn.EKS.Cluster, HostedIn.K8s.Namespace, RemoteService, RemoteOperation

Amazon EKS クラスター以外のプラットフォームで実行可能

- Service, Operation, HostedIn.Environment
- Service, HostedIn.Environment
- Service, Operation, HostedIn.Environment, RemoteService, RemoteOperation, RemoteTarget
- Service, Operation, HostedIn.Environment, RemoteService, RemoteOperation,
- Service, HostedIn.Environment, RemoteService
- Service, HostedIn.Environment, RemoteService, RemoteOperation, RemoteTarget
- Service, HostedIn.Environment, RemoteService, RemoteOperation,

合成モニタリングの使用

Amazon CloudWatch Synthetics を使用して、スケジュールに沿って実行される設定可能なスクリプトである Canary を作成し、エンドポイントと API をモニターリングできます。Canary は顧客と同じルートをたどり、同じアクションを実行します。これにより、アプリケーションに顧客トラフィックがない場合でも、顧客エクスペリエンスを継続的に検証できます。Canary を使用すると、顧客が問題を検出する前に問題を検出できます。

Canary は、Node.js または Python で記述されたスクリプトです。Node.js または Python をフレームワークとして使用する Lambda 関数をアカウント内に作成します。Canary は、HTTP プロトコルと HTTPS プロトコルの両方で動作します。Canary は、CloudWatch Synthetics ライブラリを含む Lambda レイヤーを使用します。このライブラリには、NodeJS Canary 用 CloudWatch Synthetics の NodeJS バージョンと、Python Canary 用 CloudWatch Synthetics の Python バージョンが含まれています。レイヤーは CloudWatch Synthetics サービスアカウントに属します。ライブラリが顧客情

報を転送したり保存したりすることはありません。すべての顧客データは、顧客アカウントにのみ保存されます。

Canary は、Puppeteer を介してヘッドレス Google Chrome ブラウザまたは Selenium Webdriver へのプログラムアクセスを提供します。Puppeteer の詳細については、「Puppeteer」を参照してください。Selenium の詳細については、www.selenium.dev/ をご参照ください。

Canary はエンドポイントの可用性とレイテンシーをチェックし、読み込み時間のデータと UI のスクリーンショットを保存できます。REST API、URL、ウェブサイトのコンテンツをモニターリングし、フィッシング、コードインジェクション、およびクロスサイトスクリプティングによる不正な変更をチェックできます。

CloudWatch Synthetics は Application Signals と統合されており、これらを導入することで、アプリケーションサービス、クライアント、Synthetics canary、サービスの依存関係を検出し、モニタリングできます。Application Signals を使用すると、サービスのリストやビジュアルマップを確認したり、サービスレベル目標 (SLO) に基づくヘルスメトリクスを表示したり、ドリルダウンして相関関係のある X-Ray トレースを確認したりして、より詳細なトラブルシューティングを行うことができます。Application Signals で canary を表示するには、X-Ray アクティブトレースを有効にします。canary は、サービスに関連付けした [サービスマップ] に加え、canary が呼び出すサービスの [サービス詳細] ページに表示されます。

Canary の動画デモについては、次を参照してください。

- 「Introduction to Amazon CloudWatch Synthetics」(Amazon CloudWatch Synthetics の概要)
- Amazon CloudWatch Synthetics のデモ
- 「Create Canaries Using Amazon CloudWatch Synthetics」(Amazon CloudWatch Synthetics を使用して Canary を作成する)
- 「Visual Monitoring with Amazon CloudWatch Synthetics」(Amazon CloudWatch Synthetics でのビジュアルモニターリング)

Canary は 1 回実行するか、定期的なスケジュールで実行できます。Canary は毎分 1 回の頻度で実行することができます。Canary をスケジュールする際は、cron 式とレート式のどちらも使えます。

Canary を作成および実行する前に考慮すべきセキュリティの問題については、「Synthetics Canary のセキュリティ上の考慮事項」を参照してください。

Canary は、デフォルトで `CloudWatchSynthetics` 名前空間にいくつかの CloudWatch メトリクスを作成します。これらのメトリクスには、ディメンションとして `CanaryName` があります。関数

ライブラリの `executeStep()` または `executeHttpStep()` 関数を使用する Canary にも、ディメンションとして `StepName` があります。Canary 関数ライブラリの詳細については、「Canary スクリプト用のライブラリ関数」を参照してください。

CloudWatch Synthetics と X-Ray トレースマップは効果的に連携できます。CloudWatch と AWS X-Ray の連携により、エンドツーエンドでのサービス確認、パフォーマンスボトルネック特定の効率化、影響を受けるユーザーの把握が可能になります。CloudWatch Synthetics で作成した canary は、トレースマップに表示されます。詳細については、「Using the X-Ray trace map」を参照してください。

現在 CloudWatch Synthetics は、すべての商用 AWS リージョンならびに GovCloud リージョンでご利用が可能です。

> **Note**
>
> アジアパシフィック (大阪) では、AWS PrivateLink はサポートされていません。アジアパシフィック (ジャカルタ) では、AWS PrivateLink と X-Ray はサポートされていません。

トピック

CloudWatch Canary に必要なロールとアクセス許可

Canary を作成および管理するユーザーと、Canary 自身の両方に特定の許可が必要です。

CloudWatch Canary を管理するユーザーに必要なロールと許可

Canary の詳細と Canary の実行結果を表示するには、CloudWatchSyntheticsFullAccess または CloudWatchSyntheticsReadOnlyAccess ポリシーをアタッチしたユーザーとしてサインインする必要があります。コンソールですべての Synthetics データを読み取るには、AmazonS3ReadOnlyAccess ポリシーと CloudWatchReadOnlyAccess ポリシーも必要です。Canary で使用しているソースコードを表示するには、AWSLambda_ReadOnlyAccess ポリシーも必要です。

Canary を作成するには、CloudWatchSyntheticsFullAccess ポリシーまたは同様のアクセス許可セットを持つユーザーとしてサインインする必要があります。Canary の IAM ロールを作成するには、次のインラインポリシーステートメントも必要です。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iam:CreateRole",
                "iam:CreatePolicy",
                "iam:AttachRolePolicy"
            ],
            "Resource": [
                "arn:aws:iam::*:role/service-role/CloudWatchSyntheticsRole*",
                "arn:aws:iam::*:policy/service-role/CloudWatchSyntheticsPolicy*"
            ]
        }
    ]
}
```

> **Important**
>
> ユーザーに、iam:CreateRole、iam:CreatePolicy、および iam:AttachRolePolicy アクセス許可を付与すると、そのユーザーには、AWS アカウントへの完全な管理アクセス許可が与えられます。例えば、これらのアクセス許可を持つユーザーは、すべてのリソー

スに対する完全なアクセス許可を持つポリシーを作成し、そのポリシーを任意のロールにアタッチできます。これらのアクセス許可を付与するユーザーには十分注意してください。

ポリシーのアタッチとユーザーへのアクセス許可の付与については、「IAM ユーザーのアクセス許可の変更」および「ユーザーまたはロールのインラインポリシーを埋め込むには」を参照してください。

Canary に必要なロールとアクセス許可

各 Canary は、特定の許可がアタッチされた IAM ロールに関連付ける必要があります。CloudWatch コンソールを使用して Canary を作成する場合、CloudWatch Synthetics で Canary の IAM ロールを作成することを選択できます。これを実行すると、ロールに必要な許可が付与されます。

自ら IAM ロールを作成する場合、または AWS CLI もしくは API を使用して Canary を作成するときに使用できる IAM ロールを作成する場合、そのロールにはこのセクションにリストされている許可が含まれている必要があります。

Canary のすべての IAM ロールには、次の信頼ポリシーステートメントを含める必要があります。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "lambda.amazonaws.com"
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

さらに、Canary の IAM ロールには、次のいずれかのステートメントが必要です。

AWS KMS を使用せず、Amazon VPC アクセスを必要としない基本的な Canary

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

```
            "Action": [
                "s3:PutObject",
                "s3:GetObject"
            ],
            "Resource": [
                "arn:aws:s3:::path/to/your/s3/bucket/canary/results/folder"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetBucketLocation"
            ],
            "Resource": [
                "arn:aws:s3:::name/of/the/s3/bucket/that/contains/canary/results"

            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogStream",
                "logs:PutLogEvents",
                "logs:CreateLogGroup"
            ],
            "Resource": [
                "arn:aws:logs:canary_region_name:canary_account_id:log-group:/aws/
lambda/cwsyn-canary_name-*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:ListAllMyBuckets",
                "xray:PutTraceSegments"
            ],
            "Resource": [
                "*"
            ]
        },
        {
            "Effect": "Allow",
            "Resource": "*",
            "Action": "cloudwatch:PutMetricData",
```

```
            "Condition": {
               "StringEquals": {
                "cloudwatch:namespace": "CloudWatchSynthetics"
               }
            }
        }
    ]
}
```

Canary アーティファクトの暗号化に AWS KMS を使用するが、Amazon VPC アクセスを必要としない Canary

```
{
    "Version": "2012-10-17",
    "Statement": [{
            "Effect": "Allow",
            "Action": [
                "s3:PutObject",
                "s3:GetObject"
            ],
            "Resource": [
                "arn:aws:s3:::path/to/your/S3/bucket/canary/results/folder"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetBucketLocation"
            ],
            "Resource": [
                "arn:aws:s3:::name/of/the/S3/bucket/that/contains/canary/results"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogStream",
                "logs:PutLogEvents",
                "logs:CreateLogGroup"
            ],
            "Resource": [
                "arn:aws:logs:canary_region_name:canary_account_id:log-group:/aws/
lambda/cwsyn-canary_name-*"
```

```
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:ListAllMyBuckets",
                "xray:PutTraceSegments"
            ],
            "Resource": [
                "*"
            ]
        },
        {
            "Effect": "Allow",
            "Resource": "*",
            "Action": "cloudwatch:PutMetricData",
            "Condition": {
                "StringEquals": {
                    "cloudwatch:namespace": "CloudWatchSynthetics"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "kms:Decrypt",
                "kms:GenerateDataKey"
            ],
            "Resource":
 "arn:aws:kms:KMS_key_region_name:KMS_key_account_id:key/KMS_key_id",
            "Condition": {
                "StringEquals": {
                    "kms:ViaService": [
                        "s3.region_name_of_the_canary_results_S3_bucket.amazonaws.com"
                    ]
                }
            }
        }
    ]
}
```

AWS KMS を使用しないが Amazon VPC アクセスを必要とする Canary

```
{
    "Version": "2012-10-17",
    "Statement": [{
            "Effect": "Allow",
            "Action": [
                "s3:PutObject",
                "s3:GetObject"
            ],
            "Resource": [
                "arn:aws:s3:::path/to/your/S3/bucket/canary/results/folder"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetBucketLocation"
            ],
            "Resource": [
                "arn:aws:s3:::name/of/the/S3/bucket/that/contains/canary/results"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogStream",
                "logs:PutLogEvents",
                "logs:CreateLogGroup"
            ],
            "Resource": [
                "arn:aws:logs:canary_region_name:canary_account_id:log-group:/aws/
lambda/cwsyn-canary_name-*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:ListAllMyBuckets",
                "xray:PutTraceSegments"
            ],
            "Resource": [
                "*"
            ]
        },
```

```
        {
            "Effect": "Allow",
            "Resource": "*",
            "Action": "cloudwatch:PutMetricData",
            "Condition": {
                "StringEquals": {
                    "cloudwatch:namespace": "CloudWatchSynthetics"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "ec2:CreateNetworkInterface",
                "ec2:DescribeNetworkInterfaces",
                "ec2:DeleteNetworkInterface"
            ],
            "Resource": [
                "*"
            ]
       }
    ]
}
```

Canary アーティファクトの暗号化に AWS KMS を使用し、Amazon VPC アクセスも必要とする Canary

VPC 以外の Canary を更新して VPC の使用を開始する場合は、次のポリシーにリストされているネットワークインターフェイス許可を含めるように Canary のロールを更新する必要があります。

```
{
    "Version": "2012-10-17",
    "Statement": [{
            "Effect": "Allow",
            "Action": [
                "s3:PutObject",
                "s3:GetObject"
            ],
            "Resource": [
                "arn:aws:s3:::path/to/your/S3/bucket/canary/results/folder"
            ]
        },
        {
```

```
            "Effect": "Allow",
            "Action": [
                "s3:GetBucketLocation"
            ],
            "Resource": [
                "arn:aws:s3:::name/of/the/S3/bucket/that/contains/canary/results"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogStream",
                "logs:PutLogEvents",
                "logs:CreateLogGroup"
            ],
            "Resource": [
                "arn:aws:logs:canary_region_name:canary_account_id:log-group:/aws/
lambda/cwsyn-canary_name-*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:ListAllMyBuckets",
                "xray:PutTraceSegments"
            ],
            "Resource": [
                "*"
            ]
        },
        {
            "Effect": "Allow",
            "Resource": "*",
            "Action": "cloudwatch:PutMetricData",
            "Condition": {
                "StringEquals": {
                    "cloudwatch:namespace": "CloudWatchSynthetics"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "ec2:CreateNetworkInterface",
```

```
                "ec2:DescribeNetworkInterfaces",
                "ec2:DeleteNetworkInterface"
            ],
            "Resource": [
                "*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "kms:Decrypt",
                "kms:GenerateDataKey"
            ],
            "Resource":
 "arn:aws:kms:KMS_key_region_name:KMS_key_account_id:key/KMS_key_id",
            "Condition": {
                "StringEquals": {
                    "kms:ViaService": [
                        "s3.region_name_of_the_canary_results_S3_bucket.amazonaws.com"
                    ]
                }
            }
        }
    ]
}
```

AWSCloudWatch Synthetics の マネージドポリシー

ユーザー、グループ、ロールにアクセス許可を追加するには、自分でポリシーを作成するよりも、AWS 管理ポリシーを使用する方が簡単です。チームに必要な権限のみを提供する IAM カスタマーマネージドポリシーを作成するには、時間と専門知識が必要です。すぐに使用を開始するために、AWS マネージドポリシーを使用できます。これらのポリシーは、一般的なユースケースを対象範囲に含めており、AWS アカウントで利用できます。AWS マネージドポリシーの詳細については、IAM ユーザーガイドの「AWS マネージドポリシー」AWS マネージドポリシーを参照してください。

AWS のサービスは、AWS マネージドポリシーを維持および更新します。AWS マネージドポリシーの許可を変更することはできません。サービスによって、AWS 管理ポリシーのアクセス権限が変更されることがあります。このタイプの更新は、ポリシーがアタッチされているすべての ID (ユーザー、グループ、ロール) に影響します。

AWS マネージドポリシーへの CloudWatch Synthetics の更新

このサービスがこれらの変更の追跡を開始してからの、CloudWatch Synthetics の AWS マネージドポリシーの更新に関する詳細を表示します。このページの変更に関する自動通知を入手するには、CloudWatch ドキュメントの履歴ページから、RSS フィードにサブスクライブしてください。

変更	説明	日付
CloudWatchSyntheticsFullAccess から削除された冗長アクション	`s3:PutBucketEncryption` および `lambda:GetLayerVersionByArn` アクションがポリシー内の他のアクセス権限と冗長であるため、CloudWatch Synthetics は、CloudWatchSyntheticsFullAccess ポリシーからこれらのアクションを削除しました。削除されたアクションはアクセス権限を提供しなかったため、ポリシーによって付与されたアクセス権限に差分変更はありません。	2021 年 3 月 12 日
CloudWatch Synthetics の変更の追跡を開始しました	CloudWatch Synthetics は、AWS マネージドポリシーの変更の追跡を開始しました。	2021 年 3 月 10 日

CloudWatchSyntheticsFullAccess

`CloudWatchSyntheticsFullAccess` ポリシーの内容は次のとおりです。

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

```
            "synthetics:*"
         ],
         "Resource":"*"
      },
      {
         "Effect":"Allow",
         "Action":[
            "s3:CreateBucket",
            "s3:PutEncryptionConfiguration"
         ],
         "Resource":[
            "arn:aws:s3:::cw-syn-results-*"
         ]
      },
      {
         "Effect":"Allow",
         "Action":[
            "iam:ListRoles",
            "s3:ListAllMyBuckets",
            "s3:GetBucketLocation",
            "xray:GetTraceSummaries",
            "xray:BatchGetTraces",
            "apigateway:GET"
         ],
         "Resource":"*"
      },
      {
         "Effect":"Allow",
         "Action":[
            "s3:GetObject",
            "s3:ListBucket"
         ],
         "Resource":"arn:aws:s3:::cw-syn-*"
      },
      {
         "Effect":"Allow",
         "Action":[
            "s3:GetObjectVersion"
         ],
         "Resource":"arn:aws:s3:::aws-synthetics-library-*"
      },
      {
         "Effect":"Allow",
         "Action":[
```

```
            "iam:PassRole"
         ],
         "Resource":[
            "arn:aws:iam::*:role/service-role/CloudWatchSyntheticsRole*"
         ],
         "Condition":{
            "StringEquals":{
               "iam:PassedToService":[
                  "lambda.amazonaws.com",
                  "synthetics.amazonaws.com"
               ]
            }
         }
      },
      {
         "Effect":"Allow",
         "Action":[
            "iam:GetRole"
         ],
         "Resource":[
            "arn:aws:iam::*:role/service-role/CloudWatchSyntheticsRole*"
         ]
      },
      {
         "Effect":"Allow",
         "Action":[
            "cloudwatch:GetMetricData",
            "cloudwatch:GetMetricStatistics"
         ],
         "Resource":"*"
      },
      {
         "Effect":"Allow",
         "Action":[
            "cloudwatch:PutMetricAlarm",
            "cloudwatch:DeleteAlarms"
         ],
         "Resource":[
            "arn:aws:cloudwatch:*:*:alarm:Synthetics-*"
         ]
      },
      {
         "Effect":"Allow",
         "Action":[
```

```
            "cloudwatch:DescribeAlarms"
         ],
         "Resource":[
            "arn:aws:cloudwatch:*:*:alarm:*"
         ]
      },
      {
         "Effect":"Allow",
         "Action":[
            "lambda:CreateFunction",
            "lambda:AddPermission",
            "lambda:PublishVersion",
            "lambda:UpdateFunctionConfiguration",
            "lambda:GetFunctionConfiguration"
         ],
         "Resource":[
            "arn:aws:lambda:*:*:function:cwsyn-*"
         ]
      },
      {
         "Effect":"Allow",
         "Action":[
            "lambda:GetLayerVersion",
            "lambda:PublishLayerVersion"
         ],
         "Resource":[
            "arn:aws:lambda:*:*:layer:cwsyn-*",
            "arn:aws:lambda:*:*:layer:Synthetics:*"
         ]
      },
      {
         "Effect":"Allow",
         "Action":[
            "ec2:DescribeVpcs",
            "ec2:DescribeSubnets",
            "ec2:DescribeSecurityGroups"
         ],
         "Resource":[
            "*"
         ]
      },
      {
         "Effect":"Allow",
         "Action":[
```

```
            "sns:ListTopics"
         ],
         "Resource":[
            "*"
         ]
      },
      {
         "Effect":"Allow",
         "Action":[
            "sns:CreateTopic",
            "sns:Subscribe",
            "sns:ListSubscriptionsByTopic"
         ],
         "Resource":[
            "arn:*:sns:*:*:Synthetics-*"
         ]
      }
   ]
}
```

CloudWatchSyntheticsReadOnlyAccess

`CloudWatchSyntheticsReadOnlyAccess` ポリシーの内容は次のとおりです。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "synthetics:Describe*",
                "synthetics:Get*",
                "synthetics:List*"
            ],
            "Resource": "*"
        }
    ]
}
```

特定の canary を表示するようにユーザーを限定する

canary に関する情報を表示するユーザーの機能を制限して、指定した canary に関する情報のみを表示できるようにします。これを行うには、次のような Condition ステートメントで IAM ポリシーを使用し、このポリシーをユーザーまたは IAM ロールにアタッチします。

次の例では、ユーザーが name-of-allowed-canary-1 と name-of-allowed-canary-2 の情報のみを表示するように制限します。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "synthetics:DescribeCanaries",
            "Resource": "*",
            "Condition": {
                "ForAnyValue:StringEquals": {
                    "synthetics:Names": [
                        "name-of-allowed-canary-1",
                        "name-of-allowed-canary-2"
                    ]
                }
            }
        }
    ]
}
```

CloudWatch Synthetics では、synthetics:Names 配列に最大 5 つの項目をリストできます。

次の例に示すように、許可される canary 名で * をワイルドカードとして使用するポリシーを作成することもできます。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": "synthetics:DescribeCanaries",
            "Resource": "*",
```

```
            "Condition": {
                "ForAnyValue:StringLike": {
                    "synthetics:Names": [
                        "my-team-canary-*"
                    ]
                }
            }
        }
    ]
}
```

これらのポリシーのいずれかをアタッチしてサインインしたユーザーは、CloudWatch コンソールを使用して canary 情報を表示することはできません。これらのユーザーは、ポリシーで承認された canary に関する canary 情報のみを DescribeCanaries API または describe-canaries AWS CLI コマンドを使用して表示できます。

Canary を作成する

> **Important**
>
> Synthetics Canary を使用して、所有権またはアクセス許可を持つエンドポイントと API のみを監視します。Canary の実行回数の設定によっては、これらのエンドポイントでのトラフィックが増える場合があります。

CloudWatch コンソールを使用して Canary を作成する場合、CloudWatch が提供するブループリントを使用して Canary を作成することも、独自のスクリプトを作成することもできます。詳細については、「Canary 設計図の使用」を参照してください。

Canary に独自のスクリプトを使用している場合は、AWS CloudFormation を使用して Canary を作成することもできます。詳細については、AWS CloudFormation ユーザーガイドの AWS::Synthetics::Canary を参照してください。

独自のスクリプトを作成する場合は、CloudWatch Synthetics によってライブラリに組み込み済みの複数の関数を使用できます。詳細については、「Synthetics のランタイムバージョン」を参照してください。

> **Note**
>
> Canary を作成すると、作成されるレイヤーの 1 つは、先頭に `Synthetics` の追加された Synthetics レイヤーになります。このレイヤーは Synthetics サービスアカウントが所有し、ランタイムコードを含みます。

Canary を作成するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで、[Application Signals]、[Synthetics Canary] の順に選択します。
3. [Create Canary (Canary を作成)] を選択します。
4. 次のいずれかを選択します。
   - Canary を設計図スクリプトに基づいて作成するには、[Use a blueprint] を選択し、作成する Canary のタイプを選択します。設計図のタイプ別の詳細な動作については、「Canary 設計図の使用」を参照してください。
   - 独自の Node.js スクリプトをアップロードしてカスタムの Canary を作成するには、[Upload a script] を選択します。

     次に、スクリプトを [Script] 領域にドラッグするか、[Browse files] を選択してファイルシステム内のスクリプトに移動できます。
   - S3 バケットからスクリプトをインポートするには、[Import from S3] を選択します。[Source location] で、Canary への完全なパスを入力するか、[Browse S3] を選択します。

     使用する S3 バケットの `s3:GetObject` および `s3:GetObjectVersion` アクセス許可が必要です。バケットは、Canary を作成するのと同じ AWS リージョンに存在する必要があります。
5. [名前] に、Canary の名前を入力します。この名前は多くのページで使用されるため、他の Canary と区別できるわかりやすい名前を付けてください。
6. [Application or endpoint URL] に、Canary でテストする URL を入力します。この URL には、プロトコル (https:// など) を含める必要があります。

   VPC のエンドポイントを Canary でテストする場合は、この手順の後半で VPC に関する情報も入力する必要があります。

7. Canary 用に独自のスクリプトを使用している場合は、[Lambda handler] に Canary を開始するエントリポイントを入力します。`syn-nodejs-puppeteer-3.4` または `syn-python-selenium-1.1` より前のランタイムを使用する場合、入力する文字列の末尾は `.handler` である必要があります。`syn-nodejs-puppeteer-3.4` または `syn-python-selenium-1.1` 以降のランタイムを使用する場合、この制限は適用されません。
8. スクリプトで環境変数を使用している場合は、[Environment variables] (環境変数) を選択し、スクリプトで定義されている各環境変数の値を指定します。詳細については、「環境変数」を参照してください。
9. [スケジュール] で、この Canary を 1 回だけ実行するか、レート式を使用して連続して実行するか、cron 式を使用してスケジュールするかを選択します。

   - CloudWatch コンソールを使用して継続的に動作する Canary を作成する場合、1 分から 1 時間に 1 回までの任意の間でレートを選択できます。

   - Canary スケジューリング用の cron 式の記述の詳細については、「cron を使用して Canary 実行をスケジュールする」を参照してください。
10. (オプション) canary のタイムアウト値を設定するには、[追加設定] を選択してタイムアウト値を指定します。Lambda コールドスタートと canary インストルメンテーションの起動にかかる時間を許容するには、15 秒以上にします。
11. [Data retention (データ保持)] で、失敗した Canary の実行と成功した Canary の実行の両方に関する情報を保持する期間を指定します。指定できる範囲は 1～455 日です。

    この設定は、CloudWatch Synthetics がコンソールに保存および表示するデータにのみ影響します。Amazon S3 バケットに保存されているデータ、canary によって公開されるログやメトリックスには影響しません。
12. [Data Storage] で、Canary の実行データを保存する先の S3 バケットを選択します。バケット名にピリオド (.) を含めることはできません。これを空白のままにすると、デフォルトの S3 バケットが使用または作成されます。

    `syn-nodejs-puppeteer-3.0` 以降のランタイムを使用している場合、テキストボックスにバケットの URL を入力するときに、現在のリージョンまたは別のリージョンにバケットを指定できます。以前のランタイムバージョンを使用している場合、バケットは現在のリージョンに存在する必要があります。
13. (オプション) Canary のデフォルトでは、そのアーティファクトが Amazon S3 に保存され、そのアーティファクトに対しては AWS 管理のAWS KMS キーにより保存時の暗号化が行われます。[Data Storage] (データストレージ) セクションの [Additional configuration] (その他の設定)

をクリックすることで、別の暗号化オプションを使用することも可能です。その後、暗号化に使用するキーのタイプを選択します。詳細については、「Canary アーティファクトの暗号化」を参照してください。

14. [Access permissions] で、Canary を実行するための IAM ロールを作成するか、既存のロールを使用するかを選択します。

    CloudWatch Synthetics がロールを作成している場合、必要な許可がすべて自動的に含まれます。自らロールを作成する場合は、必要な許可に関する情報について、「Canary に必要なロールとアクセス許可」を参照してください。

    Canary の作成時に CloudWatch コンソールを使用して Canary 用のロールを作成する場合、他の Canary 用のロールは再利用できません。これらのロールは Canary 別に固有です。複数の Canary 用のロールを手動で作成済みである場合は、その既存のロールを使用できます。

    既存のロールを使用するには、そのロールを Synthetics および Lambda に渡すための `iam:PassRole` アクセス許可が必要です。また、`iam:GetRole` アクセス許可も必要です。

15. (オプション) [Alarms] で、この Canary 用にデフォルトの CloudWatch アラームを作成するかどうかを選択します。アラームの作成を選択すると、`Synthetics-Alarm-`*`canaryName`*`-`*`index`* の名前規則で作成されます。

    `index` は、この Canary 用に作成された異なる各アラームを表す数値です。最初のアラームのインデックスは 1、2 番目のアラームのインデックスは 2 です。

16. (オプション) この Canary で VPC のエンドポイントをテストするには、[VPC settings] を選択し、次の操作を行います。

    a. エンドポイントをホストする VPC を選択します。

    b. VPC 上の 1 つ以上のサブネットを選択します。実行中に IP アドレスを Lambda インスタンスに割り当てることができない場合は、Lambda インスタンスをパブリックサブネットで実行するように設定できないため、プライベートサブネットを選択する必要があります。詳細については、「VPC 内のリソースにアクセスできるように Lambda 関数を設定する」を参照してください。

    c. VPC 上で 1 つ以上のセキュリティグループを選択します。

    エンドポイントが VPC 上にある場合は、Canary から CloudWatch と Amazon S3 に情報を送信できるようにする必要があります。詳細については、「VPC で Canary を実行する」を参照してください。

17. (オプション) [Tags] で、この Canary のタグとする 1 つ以上のキーと値のペアを追加します。タグを使用すると、AWS リソースを識別して整理したり、AWS コストを追跡したりしやすくなります。詳細については、「Amazon CloudWatch リソースにタグを付ける」を参照してください。
18. (オプション) [Active tracing (アクティブトレース)] で、この Canary にアクティブな X-Ray トレースを有効にするかどうかを選択します。このオプションは、Canary がランタイムバージョン syn-nodejs-2.0 以降を使用している場合にのみ使用できます。詳細については、「Canary と X-Ray のトレース」を参照してください。

Canary 向けに作成されたリソース

Canary を作成すると、次のリソースが作成されます。

- `CloudWatchSyntheticsRole-`*`canary-name`*`-`*`uuid`* という名前の IAM ロール (CloudWatch コンソールを使用して Canary を作成し、この Canary 用に新しいロールを作成することを指定した場合)
- `CloudWatchSyntheticsPolicy-`*`canary-name`*`-`*`uuid`* という名前の IAM ポリシー
- `cw-syn-results-`*`accountID`*`-`*`region`* という名前の S3 バケット
- `Synthetics-Alarm-`*`MyCanaryName`* という名前のアラーム (Canary 用にアラームを作成した場合)
- Lambda 関数とレイヤー (設計図を使用して Canary を作成した場合)。これらのリソースには、プレフィックスとして `cwsyn-`*`MyCanaryName`* が付きます。
- `/aws/lambda/cwsyn-`*`MyCanaryName`*`-`*`randomId`* という名前の CloudWatch Logs ロググループ。

Canary 設計図の使用

このセクションでは、Canary の各設計図と、各設計図に最も適合するタスクについて詳しく説明します。設計図は、以下の Canary タイプのために提供されています:

- ハートビートモニター
- API Canary
- リンク切れチェッカー
- ビジュアルモニターリング
- Canary Recorder

- GUI ワークフロー

設計図を使用して Canary を作成する場合、CloudWatch コンソールの各フィールドに入力するに従って、作成中の Canary がページの [スクリプトエディタ] 領域に Node.js スクリプトとして表示されます。この領域で Canary を編集し、さらにカスタマイズすることもできます。

ハートビートのモニターリング

ハートビートスクリプトは、指定した URL をロードし、ページのスクリーンショットと HTTP アーカイブファイル (HAR ファイル) を保存します。また、アクセスした URL のログも保存します。

HAR ファイルを使用して、ウェブページに関する詳細なパフォーマンスデータを表示できます。ウェブリクエストのリストを分析し、項目の読み込み時間などのパフォーマンス問題を検出できます。

Canary が `syn-nodejs-puppeteer-3.1` 以降のランタイムバージョンを使用している場合、ハートビートモニターリングブループリントを使用して複数の URL をモニターリングし、Canary 実行レポートのステップの概要で各 URL のステータス、期間、関連付けられたスクリーンショット、失敗の理由を確認できます。

API Canary

API Canary は、REST API の基本的な読み取り関数と書き込み関数をテストできます。RESTは Representational State Transfer の略であり、開発者が API を作成するときに従う一連のルールです。これらのルールの 1 つでは、特定の URL へのリンクがデータを返す必要があることを指定しています。

Canary は任意の API で動作し、すべてのタイプの機能をテストできます。各 Canary は複数の API 呼び出しを行うことができます。

ランタイムバージョン `syn-nodejs-2.2` 以降を使用する Canary では、API Canary ブループリントは、API を HTTP ステップとしてモニターリングする複数ステップの Canary をサポートします。1 つの Canary で複数の API をテストできます。各ステップは、異なる URL にアクセスしたり、異なるヘッダーを使用したり、ヘッダーとレスポンス本文を取得するかどうかについて異なるルールを使用したりできる個別のリクエストです。ヘッダーとレスポンス本文を取得しないことで、機密データが記録されないようにできます。

API Canary 内の各リクエストは、次の情報で構成されます。

- エンドポイント。リクエストする URL です。

- メソッド。サーバーに送信されるリクエストのタイプです。REST API は、GET (読み取り)、POST (書き込み)、PUT (更新)、PATCH (更新)、DELETE (削除) の各オペレーションをサポートしています。
- ヘッダー。クライアントとサーバーの両方に情報を提供します。これらは認証に使用され、本文の内容に関する情報を提供します。有効なヘッダーのリストについては、「HTTPヘッダー」をご参照ください。
- データ (または本文)。サーバーに送信される情報を含みます。これは、POST、PUT、PATCH、または DELETE の各リクエストでのみ使用されます。

API Canary 設計図は、GET メソッドと POST メソッドをサポートしています。この設計図を使用する場合は、ヘッダーを指定する必要があります。例えば、**`Authorization`** をキーとして指定し、必要な認証データを値として指定できます。

POST リクエストをテストする場合は、[データ] フィールドに書き込むコンテンツも指定します。

API Gateway との統合

API ブループリントは Amazon API Gateway と統合されています。これにより、API Gateway API やステージを Canary と同じ AWS アカウントやリージョンから選択したり、API Gateway から Swagger テンプレートをアップロードして、クロスアカウントとクロスリージョン API のモニターリングを行ったりすることができます。その後、最初から入力するのではなく、コンソールで残りの詳細を選択して Canary を作成できます。API Gateway の詳細については、「Amazon API Gateway とは」を参照してください。

プライベート API の使用

Amazon API Gateway でプライベート API を使用する Canary を作成できます。詳細については、「Amazon API Gateway でのプライベート API の作成」を参照してください。

リンク切れチェッカー

リンク切れチェッカーは、`document.getElementsByTagName('a')` を使用して、テスト対象である URL 内のすべてのリンクを収集します。この際、ユーザーが指定したリンク数までがテストされ、また、URL 自体も最初のリンクとしてカウントされます。例えば、5 つのリンクが含まれているページのすべてのリンクをチェックする場合は、Canary が 6 つのリンクをたどるように指定する必要があります。

`syn-nodejs-2.0-beta` ランタイム以降を使用して作成されたリンク切れチェッカー Canary は、次の追加機能をサポートしています。

- チェックされたリンク、ステータスコード、エラーの理由 (存在する場合)、およびソースページとターゲットページのスクリーンショットを含むレポートを生成する。
- Canary の結果を表示するとき、フィルタリングしてリンク切れのみを表示し、エラーの理由に基づいてリンクを修正できます。
- このバージョンでは、各リンクの注釈付きソースページのスクリーンショットがキャプチャされ、リンクが見つかったアンカーが強調表示されます。非表示のコンポーネントには注釈が付けられません。
- キャプチャするスクリーンショットは、ソースおよびターゲットページの両方、ソースページのみ、またはターゲットページのみに設定できます。
- 最初のページから多くのリンクがスクレイプされた場合でも最初のリンク切れの後に Canary スクリプトが停止するという、以前のバージョンの問題は、このバージョンで修正されています。

新しいランタイムを導入するために、`syn-1.0` を使用している既存の Canary を更新する場合は、一度 Canary を削除した上で再作成する必要があります。既存の Canary を新しいランタイムに更新しても、これらの機能は利用可能になりません。

リンク切れチェッカー Canary は、次の種類のリンクエラーを検出します。

- 404 ページが見つからない。
- ホスト名が無効である。
- URL が正しくない。この例としては、URL から角括弧が抜けている、スラッシュが余分についている、プロトコルが間違っているなどが挙げられます。
- HTTP レスポンスコードが無効である。
- ホストサーバーが、コンテンツもなく、レスポンスコードもない、空のレスポンスを返す。
- HTTPリクエストが Canary の実行中に常にタイムアウトする。
- ホストが正しく設定されていないか、ビジーすぎるために接続が常に切断される。

ビジュアルモニターリングの設計図

ビジュアルモニターリングの設計図には、Canary 実行中に撮影されたスクリーンショットと、ベースライン Canary 実行中に撮影されたスクリーンショットを比較するコードが含まれています。2 つのスクリーンショット間の不一致がしきい値のパーセンテージを超えている場合、Canary は失敗します。ビジュアルモニターリングは、syn-puppeteer-node-3.2 以降で実行中の Canary でサポートされています。Python と Selenium を実行している Canary では現在サポートされていません。

ビジュアルモニターリングの設計図には、デフォルトの Canary 設計図スクリプトに次のコード行が含まれており、ビジュアルモニターリングが有効になります。

```
syntheticsConfiguration.withVisualCompareWithBaseRun(true);
```

この行がスクリプトに追加された後に初めて Canary が正常に実行されると、その実行中に撮影されたスクリーンショットが比較のベースラインとして使用されます。最初に Canary を実行した後、CloudWatch コンソールを使用して Canary を編集して、次のいずれかの操作を行うことができます。

- Canary の次の実行を新しいベースラインとして設定する。
- 現在のベースラインスクリーンショットに境界を描画し、ビジュアル比較時に無視するスクリーンショットの領域を指定する。
- スクリーンショットをビジュアルモニターリングに使用しないようにする。

CloudWatch コンソールを使用して Canary を編集する方法の詳細については、「Canary を編集または削除する」を参照してください。

また、`nextrun` または `lastrun` パラメーターを指定するか、Canary 実行 ID を UpdateCanary API で指定することにより、ベースラインとして使用される Canary 実行を変更することもできます。

ビジュアルモニターリングの設計図を使用する場合は、スクリーンショットを撮影する URL を入力し、差分しきい値をパーセンテージで指定します。ベースライン実行後、そのしきい値よりも大きい視覚差を検出する Canary を今後実行すると、Canary 障害がトリガーされます。ベースライン実行後、Canary を編集して、ビジュアルモニターリング中に無視するベースラインスクリーンショットに境界線を「描画」することもできます。

ビジュアルモニターリング機能は、ImageMagick オープンソースソフトウェアツールキットによって機能します。詳細については、「ImageMagick」を参照してください。

Canary Recorder

Canary レコーダーの設計図を使用すると、CloudWatch Synthetics Recorder を使用してウェブサイトでクリックおよび入力のアクションを記録し、同じステップに従う Canary を作成するために使用できる Node.js スクリプトを自動的に生成できます。CloudWatch Synthetics Recorder は、Amazon が提供する Google Chrome 拡張機能です。

クレジット: CloudWatch Synthetics Recorder は、Headless レコーダーに基づいています。

詳細については、「Google Chrome の CloudWatch Synthetics Recorder を使用する」を参照してください。

GUI ワークフロービルダー

GUI ワークフロービルダー設計図は、ウェブページに対してアクションを実行できることを検証します。例えば、ウェブページにログインフォームがある場合、Canary はユーザーフィールドとパスワードフィールドに入力し、このフォームを送信してウェブページが正しく動作していることを検証できます。

このタイプの Canary を設計図を従って作成する場合は、Canary がウェブページに対して実行するアクションを指定します。使用できるアクションは次のとおりです。

- クリック – 指定した要素を選択し、ユーザーによる要素のクリックまたは選択をシミュレートします。

  Node.js スクリプトで要素を指定するには、`[id=]` または `a[class=]` を使用します。

  Python スクリプトで要素を指定するには、`xpath //*[@id=]` または `//*[@class=]` を使用します。
- セレクタの検証 – 指定した要素がウェブページに存在することを検証します。このテストは、以前のアクションによって正しい要素がページに入力されていることを検証するのに役立ちます。

  Node.js スクリプトで検証する要素を指定するには、`[id=]` または `a[class=]` を使用します。

  Python スクリプトで検証する要素を指定するには、`xpath //*[@id=]` または `//*[class=]` を使用します。
- テキストの検証 – 指定した文字列がターゲット要素内に含まれていることを検証します。このテストは、事前のアクションが正しいテキストを表示したかどうかを確認するのに役立ちます。

  このアクションでは Puppeteer の `div[@id=]//h1` 関数を使用するため、Node.js スクリプト内の要素を指定するには `waitForXPath` などの形式を使用します。

  Python スクリプトで要素を指定するには、このアクションは Selenium で `//*[@id=]` 関数を使用するため、`implicitly_wait` または //*[@class=] などの xpath 形式を使用します。
- テキストの入力 – 指定したテキストをターゲット要素に書き込みます。

  Node.js スクリプトで検証する要素を指定するには、`[id=]` または `a[class=]` を使用します。

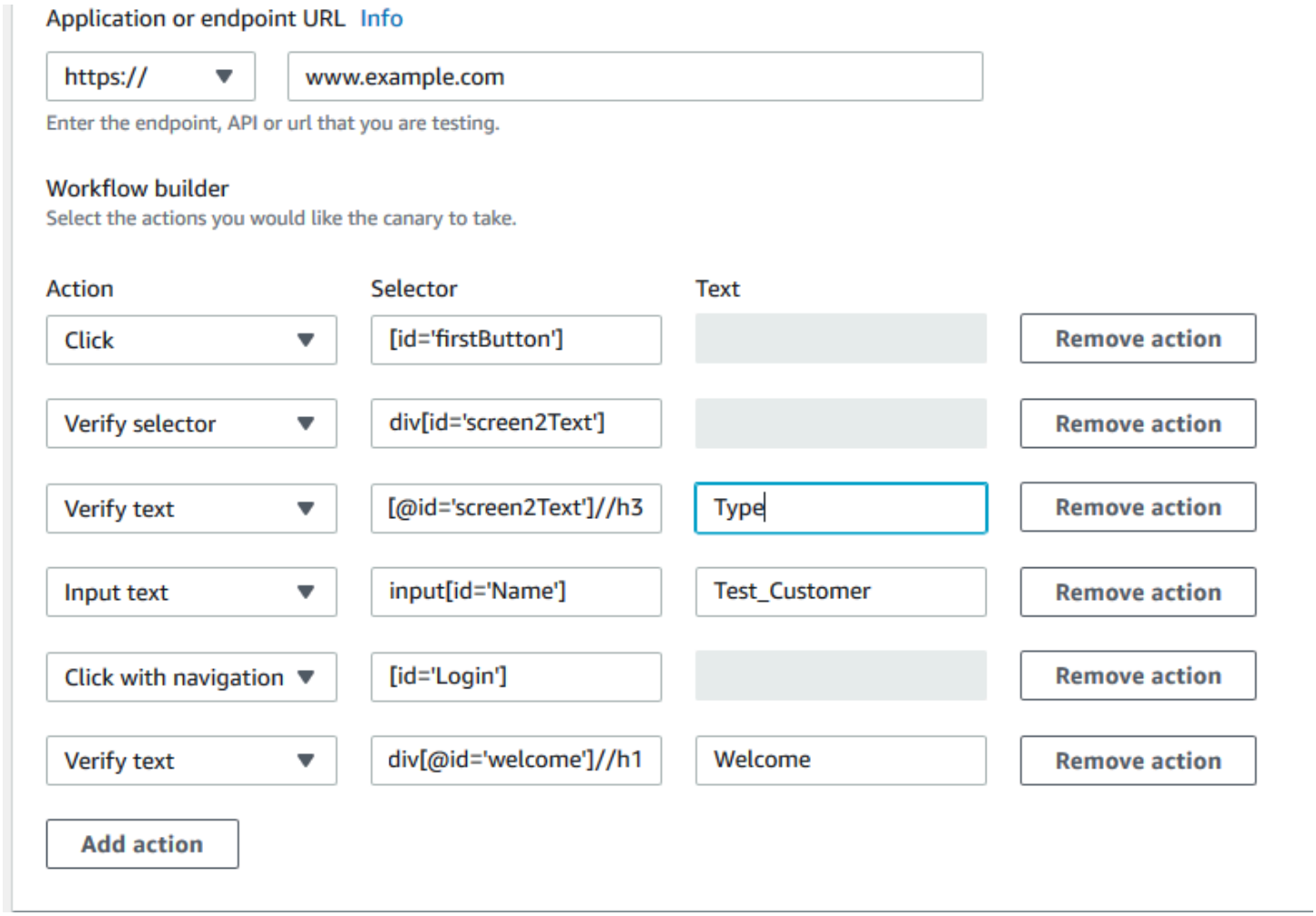
Python スクリプトで検証する要素を指定するには、`xpath //*[@id=]` または `//*[@class=]` を使用します。

- クリックして移動 – 指定した要素を選択した後で、ページ全体が読み込まれるまで待ちます。これは、ページを再度読み込む必要がある場合に最も役立ちます。

  Node.js スクリプトで要素を指定するには、`[id=]` または `a[class=]` を使用します。

  Python スクリプトで要素を指定するには、`xpath //*[@id=]` または `//*[@class=]` を使用します。

例えば、次のブループリントでは Node.js を使用しています。このスクリプトは指定した URL の [firstButton] をクリックし、想定したセレクタで想定されたテキストが表示されることを検証します。さらに、名前 `Test_Customer` を [Name] (名前) フィールドに入力して [Login] (ログイン) ボタンをクリックし、次のページに [Welcome] (ようこそ) テキストが表示されることを確認し、その口グインが成功したことを検証します。



次のランタイムを使用する GUI ワークフロー Canary は、各 Canary 実行に対して実行されたステップの概要も提供します。各ステップに関連付けられたスクリーンショットおよびエラーメッセージを使用して、エラーの根本原因を見つけることができます。

- `syn-nodejs-2.0` 以降
- `syn-python-selenium-1.0` 以降

Google Chrome の CloudWatch Synthetics Recorder を使用する

Amazon では、Canary をより簡単に作成できるように、CloudWatch Synthetics Recorder を提供しています。このレコーダーは Google Chrome の拡張機能です。

レコーダーは、ウェブサイト上のクリックと入力のアクションを記録し、同じステップに従う Canary を作成するために使用できる Node.js スクリプトを自動的に生成します。

記録を開始すると、CloudWatch Synthetics Recorder がブラウザーでのアクションを検出し、スクリプトに変換します。記録は、必要に応じて、一時停止および再開できます。記録を停止すると、レコーダーによってアクションの Node.js スクリプトが生成されます。このスクリプトは、[Copy to Clipboard (クリップボードにコピー)] ボタンで簡単にコピーできます。その後、このスクリプトを使用して、CloudWatch Synthetics で Canary を作成できます。

クレジット: CloudWatch Synthetics Recorder は、Headless レコーダーに基づいています。

Google Chrome 用 CloudWatch Synthetics Recorder 拡張機能をインストールする

CloudWatch Synthetics レコーダーを使用するには、Canary の作成を開始し、Canary Recorder の設計図を選択します。レコーダーをまだダウンロードしていない場合は、CloudWatch Synthetics コンソールにダウンロードするためのリンクが表示されます。

または、以下のステップに従って、レコーダーを直接ダウンロードしてインストールすることもできます。

CloudWatch Synthetics Recorder をインストールするには

1. Google Chromeを使用して、ウェブサイト https://chrome.google.com/webstore/detail/cloudwatch-synthetics-rec/bhdnlmmgiplmbcdmkkdfplenecpegfno にアクセスします
2. [Add to Chrome (Chrome に追加)] を選択し、[Add extension (拡張機能の追加)] を選択します。

Google Chrome の CloudWatch Synthetics Recorder を使用する

CloudWatch Synthetics Recorder を使用してCanary を作成する場合、CloudWatch コンソールで [Create canary] (Canary を作成) を選択してから、[Use a blueprint] (設計図の使用)、[Canary Recorder] を選択します。詳細については、「Canary を作成する」を参照してください。

また、レコーダーを使用してステップを記録し、すぐに Canary の作成に使用しないこともできます。

CloudWatch Synthetics Recorder を使用してウェブサイトでのアクションを記録するには

1. モニターリングするページに移動します。
2. Chrome 拡張機能のアイコンを選択し、[CloudWatch Synthetics Recorder] を選択します。
3. [Start Recording (記録の開始)] を選択します。
4. 記録するステップを実行します。録音を一時停止するには、[Pause (一時停止)] を選択します。
5. ワークフローの記録が完了したら、[Stop recording (記録の停止)] を選択します。
6. [Copy to clipboard (クリップボードにコピー)] を選択して、生成されたスクリプトをクリップボードにコピーします。または、最初からやり直す場合は、[New recording (新規録音)] を選択します。
7. コピーしたスクリプトを使用して Canary を作成するには、コピーしたスクリプトをレコーダー設計図のインラインエディタに貼り付けるか、Amazon S3 バケットに保存してそこからインポートします。
8. Canary をすぐに作成しない場合は、記録したスクリプトをファイルに保存できます。

CloudWatch Synthetics Recorder の既知の制限事項

Google Chrome 用 CloudWatch Synthetics Recorder には、現在、以下の制限があります。

- ID を持たない HTML 要素は、CSS セレクターを使用します。これは、後でウェブページの構造が変更された場合、Canary を壊す可能性があります。将来のバージョンのレコーダーでは、これに関するいくつかの設定オプション (data-id の使用など) を提供する予定です。
- レコーダーは、ダブルクリックやコピー/貼り付けなどのアクションをサポートしておらず、また、CMD+0 などのキーの組み合わせもサポートしていません。
- ページに要素またはテキストが存在することを確認するには、ユーザーがスクリプトの生成後にアサーションを追加する必要があります。レコーダーでは、その要素に対してアクションを実行しない要素の検証をサポートしていません。これは、Canary ワークフロービルダーの「テキストの確

認」または「要素の確認」オプションと同様です。私たちは、レコーダーの将来のバージョンでいくつかのアサーションサポートを追加する予定です。

- レコーダーは、記録が開始されたタブ内のすべてのアクションを記録します。ポップアップ (位置情報の追跡を許可する場合など) やポップアップから別のページへのナビゲーションは記録されません。

Synthetics のランタイムバージョン

Canary を作成または更新する際に、Canary の Synthetics ランタイムバージョンを選択します。Synthetics ランタイムは、スクリプトハンドラーを呼び出す Synthetics コードと、バンドルされた依存関係の Lambda レイヤーを組み合わせたものです。

CloudWatch Synthetics は現在、スクリプトと Puppeteer フレームワークに Node.js を使用するランタイムと、スクリプトに Python を使用し、フレームワークには Selenium Webdriver を使用するランタイムをサポートしています。

Synthetics ライブラリの最新の機能や更新プログラムを使用できるように、Canary の最新のランタイムバージョンを常に使用することをお勧めします。

Canary を作成すると、作成されるレイヤーの 1 つは、先頭に `Synthetics` の追加された Synthetics レイヤーになります。このレイヤーは Synthetics サービスアカウントが所有し、ランタイムコードを含みます。

> **Note**
>
> Canary をアップグレードして新しいバージョンの Synthetics ランタイムを使用するたびに、Canary が使用するすべての Synthetics ライブラリ関数も Synthetics ランタイムがサポートするのと同じバージョンの NodeJS に自動的にアップグレードされます。

トピック

- CloudWatch Synthetics ランタイムサポートポリシー
- Node.js と Puppeteer を使用するランタイムバージョン
- Python と Selenium Webdriver を使用するランタイムバージョン

CloudWatch Synthetics ランタイムサポートポリシー

Synthetics ランタイムバージョンは、メンテナンスおよびセキュリティ更新プログラムに依存します。ランタイムバージョンのいずれかのコンポーネントがサポートされなくなると、その Synthetics ランタイムバージョンは非推奨となります。

非推奨のランタイムバージョンを使用して Canary を作成することはできません。非推奨のランタイムを使用する Canary が引き続き実行されます。これらの Canary は、停止、開始、削除することができます。サポートされているランタイムバージョンを使用するように Canary を更新することで、非推奨のランタイムバージョンを使用する既存の Canary を更新できます。

60 日以内に非推奨となる予定のランタイムを使用している Canary がある場合、CloudWatch Synthetics は通知メールをユーザーに送信します。最新のリリースに含まれる新しい機能、セキュリティ、および強化されたパフォーマンスを活用するために、Canary のランタイムは、サポートされているバージョンに更新することをお勧めします。

Canary を新しいランタイムバージョンに更新するにはどうすればよいですか？

CloudWatch コンソールから、AWS CloudFormation、AWS CLI、または AWS SDK を使用して、Canary のランタイムバージョンを更新できます。CloudWatch コンソールを使用する場合であれば、一度に最大 5 つの Canary を更新できます。これには、一覧ページから対象の Canary を選択した上で、[アクション]、[ランタイムの更新] の順にクリックします。

アップグレードを確認するためには、CloudWatch コンソールを使用して Canary をクローンし、そのランタイムバージョンを更新します。これにより、元の Canary のクローンである別の Canary が作成されます。新しいランタイムバージョンの Canary を検証し終えたら、元の Canary のランタイムバージョンを更新した後で、クローンの Canary を削除します。

アップグレード用のスクリプトを使用すると、複数の Canary を更新することもできます。詳細については、「Canary ランタイムアップグレード用スクリプト」を参照してください。

Canary のアップグレードに失敗した場合は、「失敗した Canary のトラブルシューティング」を参照してください。

ランタイムが非推奨となる日付

ランタイムバージョン	廃止日
syn-nodejs-puppeteer-6.1	2024 年 3 月 8 日
syn-nodejs-puppeteer-6.0	2024 年 3 月 8 日
syn-nodejs-puppeteer-5.1	2024 年 3 月 8 日
syn-nodejs-puppeteer-5.0	2024 年 3 月 8 日
syn-nodejs-puppeteer-4.0	2024 年 3 月 8 日
syn-nodejs-puppeteer-3.9	2024 年 1 月 8 日
syn-nodejs-puppeteer-3.8	2024 年 1 月 8 日
syn-python-selenium-2.0	2024 年 3 月 8 日
syn-python-selenium-1.3	2024 年 3 月 8 日
syn-python-selenium-1.2	2024 年 3 月 8 日
syn-python-selenium-1.1	2024 年 3 月 8 日
syn-python-selenium-1.0	2024 年 3 月 8 日

ランタイムバージョン	廃止日
syn-nodejs-puppeteer-3.7	2024 年 1 月 8 日
syn-nodejs-puppeteer-3.6	2024 年 1 月 8 日
syn-nodejs-puppeteer-3.5	2024 年 1 月 8 日
syn-nodejs-puppeteer-3.4	2022 年 11 月 13 日
syn-nodejs-puppeteer-3.3	2022 年 11 月 13 日
syn-nodejs-puppeteer-3.2	2022 年 11 月 13 日
syn-nodejs-puppeteer-3.1	2022 年 11 月 13 日
syn-nodejs-puppeteer-3.0	2022 年 11 月 13 日
syn-nodejs-2.2	2021 年 5 月 28 日
syn-nodejs-2.1	2021 年 5 月 28 日
syn-nodejs-2.0	2021 年 5 月 28 日
syn-nodejs-2.0-beta	2021 年 2 月 8 日
syn-1.0	2021 年 5 月 28 日

Canary ランタイムアップグレード用スクリプト

Canary スクリプトを、サポートされているランタイムバージョンにアップグレードするには、次のスクリプトを使用します。

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

// You need to configure your AWS credentials and Region.
//   https://docs.aws.amazon.com/sdk-for-javascript/v3/developer-guide/setting-
credentials-node.html
//   https://docs.aws.amazon.com/sdk-for-javascript/v3/developer-guide/setting-
region.html

const synthetics = new AWS.Synthetics();

const DEFAULT_OPTIONS = {
  /**
   * The number of canaries to upgrade during a single run of this script.
   */
  count: 10,
  /**
   * No canaries are upgraded unless force is specified.
   */
  force: false
};

/**
 * The number of milliseconds to sleep between GetCanary calls when
 * verifying that an update succeeded.
 */
const SLEEP_TIME = 5000;

(async () => {
  try {
    const options = getOptions();

    const versions = await getRuntimeVersions();
    const canaries = await getAllCanaries();
    const upgrades = canaries
      .filter(canary => !versions.isLatestVersion(canary.RuntimeVersion))
      .map(canary => {
        return {
          Name: canary.Name,
          FromVersion: canary.RuntimeVersion,
```

```
          ToVersion: versions.getLatestVersion(canary.RuntimeVersion)
        };
      });

    if (options.force) {
      const promises = [];

      for (const upgrade of upgrades.slice(0, options.count)) {
        const promise = upgradeCanary(upgrade);
        promises.push(promise);
        // Sleep for 100 milliseconds to avoid throttling.
        await usleep(100);
      }

      const succeeded = [];
      const failed = [];
      for (let i = 0; i < upgrades.slice(0, options.count).length; i++) {
        const upgrade = upgrades[i];
        const promise = promises[i];
        try {
          await promise;
          console.log(`The update of ${upgrade.Name} succeeded.`);
          succeeded.push(upgrade.Name);
        } catch (e) {
          console.log(`The update of ${upgrade.Name} failed with error: ${e}`);
          failed.push({
            Name: upgrade.Name,
            Reason: e
          });
        }
      }

      if (succeeded.length) {
        console.group('The following canaries were upgraded successfully.');
        for (const name of succeeded) {
          console.log(name);
        }
        console.groupEnd()
      } else {
        console.log('No canaries were upgraded successfully.');
      }

      if (failed.length) {
        console.group('The following canaries were not upgraded successfully.');
```

```
        for (const failure of failed) {
          console.log('\x1b[31m', `${failure.Name}: ${failure.Reason}`, '\x1b[0m');
        }
        console.groupEnd();
      }
    } else {
      console.log('Run with --force [--count <count>] to perform the first <count>
 upgrades shown. The default value of <count> is 10.')
      console.table(upgrades);
    }
  } catch (e) {
    console.error(e);
  }
})();

function getOptions() {
  const force = getFlag('--force', DEFAULT_OPTIONS.force);
  const count = getOption('--count', DEFAULT_OPTIONS.count);
  return { force, count };

  function getFlag(key, defaultValue) {
    return process.argv.includes(key) || defaultValue;
  }
  function getOption(key, defaultValue) {
    const index = process.argv.indexOf(key);
    if (index < 0) {
      return defaultValue;
    }
    const value = process.argv[index + 1];
    if (typeof value === 'undefined' || value.startsWith('-')) {
      throw `The ${key} option requires a value.`;
    }
    return value;
  }
}

function getAllCanaries() {
  return new Promise((resolve, reject) => {
    const canaries = [];

    synthetics.describeCanaries().eachPage((err, data) => {
      if (err) {
        reject(err);
      } else {
```

```
        if (data === null) {
          resolve(canaries);
        } else {
          canaries.push(...data.Canaries);
        }
      }
    });
  });
}

function getRuntimeVersions() {
  return new Promise((resolve, reject) => {
    const jsVersions = [];
    const pythonVersions = [];
    synthetics.describeRuntimeVersions().eachPage((err, data) => {
      if (err) {
        reject(err);
      } else {
        if (data === null) {
          jsVersions.sort((a, b) => a.ReleaseDate - b.ReleaseDate);
          pythonVersions.sort((a, b) => a.ReleaseDate - b.ReleaseDate);
          resolve({
            isLatestVersion(version) {
              const latest = this.getLatestVersion(version);
              return latest === version;
            },
            getLatestVersion(version) {
              if (jsVersions.some(v => v.VersionName === version)) {
                return jsVersions[jsVersions.length - 1].VersionName;
              } else if (pythonVersions.some(v => v.VersionName === version)) {
                return pythonVersions[pythonVersions.length - 1].VersionName;
              } else {
                throw Error(`Unknown version ${version}`);
              }
            }
          });
        } else {
          for (const version of data.RuntimeVersions) {
            if (version.VersionName === 'syn-1.0') {
              jsVersions.push(version);
            } else if (version.VersionName.startsWith('syn-nodejs-2.')) {
              jsVersions.push(version);
            } else if (version.VersionName.startsWith('syn-nodejs-puppeteer-')) {
              jsVersions.push(version);
```

```
            } else if (version.VersionName.startsWith('syn-python-selenium-')) {
              pythonVersions.push(version);
            } else {
              throw Error(`Unknown version ${version.VersionName}`);
            }
          }
        }
      }
    });
  });
}

async function upgradeCanary(upgrade) {
  console.log(`Upgrading canary ${upgrade.Name} from ${upgrade.FromVersion} to
 ${upgrade.ToVersion}`);
  await synthetics.updateCanary({ Name: upgrade.Name, RuntimeVersion:
 upgrade.ToVersion }).promise();
  while (true) {
    await usleep(SLEEP_TIME);
    console.log(`Getting the state of canary ${upgrade.Name}`);
    const response = await synthetics.getCanary({ Name: upgrade.Name }).promise();
    const state = response.Canary.Status.State;
    console.log(`The state of canary ${upgrade.Name} is ${state}`);
    if (state === 'ERROR' || response.Canary.Status.StateReason) {
      throw response.Canary.Status.StateReason;
    }
    if (state !== 'UPDATING') {
      return;
    }
  }
}

function usleep(ms) {
  return new Promise(resolve => setTimeout(resolve, ms));
}
```

Node.js と Puppeteer を使用するランタイムバージョン

Node.js と Puppeteer の最初のランタイムバージョンには、syn-1.0 という名前が付けられました。後のランタイムバージョンには、命名規則 syn-*language*-*majorversion*.*minorversion* があります。syn-nodejs-puppeteer-3.0 以降の命名規則は syn-*language*-*framework*-*majorversion*.*minorversion* です

追加の -beta サフィックスは、ランタイムバージョンが現在ベータプレビューリリースであることを示しています。

同じメジャーバージョン番号を持つランタイムバージョンには下位互換性があります。

> **Important**
>
> 次の CloudWatch Synthetics ランタイムバージョンは 2024 年 3 月 8 日に廃止される予定です。
>
> - syn-nodejs-puppeteer-6.1
> - syn-nodejs-puppeteer-6.0
> - syn-nodejs-puppeteer-5.1
> - syn-nodejs-puppeteer-5.0
> - syn-nodejs-puppeteer-4.0
>
> 詳細については、「CloudWatch Synthetics ランタイムサポートポリシー」を参照してください。

> **Important**
>
> 重要: 付属する AWS SDK for JavaScript v2 の依存関係は削除され、将来のランタイムリリースでは AWS SDK for JavaScript v3 を使用するように更新されます。その場合は、canary コードリファレンスを更新します。または、付属する AWS SDK for JavaScript v2 依存関係をソースコードの zip ファイルに依存関係として追加することで、引き続き参照して使用することもできます。

すべてのランタイムバージョンに関する注意事項

syn-nodejs-puppeteer-3.0 ランタイムバージョンを使用する場合は、Canary スクリプトが Node.js 12.x と互換性があることを確認してください。syn-nodejs ランタイムバージョンの以前のバージョンを使用する場合は、スクリプトが Node.js 10.x と互換性があることを確認してください。

Canary 内の Lambda コードは、最大メモリが 1 GB になるように設定されています。Canary の各実行は、設定されたタイムアウト値が経過するとタイムアウトします。Canary のタイムアウト値を指

定しないと、CloudWatch は Canary の更新頻度に基づいてタイムアウト値を選択します。タイムアウト値を設定する場合、Lambda コールドスタートと canary インストルメンテーションの起動にかかる時間を許容するために 15 秒以上にします。

> **Note**
>
> 以下の CloudWatch Synthetics ランタイムバージョンは 2024 年 1 月 8 日に廃止されました。これは、2023 年 12 月 4 日に AWS Lambda が Lambda Node.js 14 ランタイムを廃止したためです。
>
> - `syn-nodejs-puppeteer-3.9`
> - `syn-nodejs-puppeteer-3.8`
> - `syn-nodejs-puppeteer-3.7`
> - `syn-nodejs-puppeteer-3.6`
> - `syn-nodejs-puppeteer-3.5`
>
> 以下の CloudWatch Synthetics ランタイムバージョンは 2022 年 11 月 13 日に廃止されました。これは、2022 年 11 月 14 日に AWS Lambda が Lambda Node.js 12 ランタイムを廃止したためです。
>
> - `syn-nodejs-puppeteer-3.4`
> - `syn-nodejs-puppeteer-3.3`
> - `syn-nodejs-puppeteer-3.2`
> - `syn-nodejs-puppeteer-3.1`
> - `syn-nodejs-puppeteer-3.0`
>
> 詳細については、「CloudWatch Synthetics ランタイムサポートポリシー」を参照してください。

syn-nodejs-puppeteer-7.0

`syn-nodejs-puppeteer-7.0` ランタイムは Lambda ランタイム Node.js 18.x の最新のランタイムバージョンです。このランタイムは Node.js と Puppeteer を使用します。

主な依存関係:

- Lambda ランタイム Node.js 18.x
- Puppeteer-core バージョン 21.9.0
- Chromium バージョン 121.0.6167.139

コードサイズ

このランタイムにパッケージ化できるコードと依存関係のサイズは 80 MB です。

syn-nodejs-puppeteer-7.0 の新機能:

- Puppeteer と Chromium のバンドルされたライブラリのバージョンを更新 — Puppeteer と Chromium の依存関係が新しいバージョンに更新されました。

> **Important**
>
> Puppeteer 19.7.0 から Puppeteer 21.9.0 に移行すると、テストとフィルターに関する重大な変更が導入されます。詳細については、「puppeteer: v20.0.0」および「puppeteer-core: v21.0.0」の「BREAKING CHANGES」セクションを参照してください。

AWS SDK v3 への推奨アップグレード

Lambda nodejs18.x ランタイムは AWS SDK v2 をサポートしていません。AWS SDK v3 に移行することを強くお勧めします。

syn-nodejs-puppeteer-6.2

主な依存関係:

- Lambda ランタイム Node.js 18.x
- Puppeteer-core バージョン 19.7.0
- Chromium バージョン 111.0.5563.146

syn-nodejs-puppeteer-6.2 の新機能:

- Chromium でバンドルされたライブラリの最新バージョン
- エフェメラルストレージモニタリング — このランタイムは、カスタマーアカウントにエフェメラルストレージモニタリングを追加します。

- バグ修正

syn-nodejs-puppeteer-5.2

`syn-nodejs-puppeteer-5.2` ランタイムは Lambda ランタイム Node.js 16.x の最新のランタイムバージョンです。このランタイムは Node.js と Puppeteer を使用します。

主な依存関係:

- Lambda ランタイム Node.js 16.x
- Puppeteer-core バージョン 19.7.0
- Chromium バージョン 111.0.5563.146

syn-nodejs-puppeteer-5.2 の新機能:

- Chromium でバンドルされたライブラリの最新バージョン
- バグ修正

syn-nodejs-puppeteer-6.1

> **Important**
>
> このランタイムバージョンは 2024 年 3 月 8 日に非推奨となる予定です。詳細については、「CloudWatch Synthetics ランタイムサポートポリシー」を参照してください。

主な依存関係:

- Lambda ランタイム Node.js 18.x
- Puppeteer-core バージョン 19.7.0
- Chromium バージョン 111.0.5563.146

syn-nodejs-puppeteer-6.1 の新機能:

- 安定性の向上 - 自動再試行ロジックを追加し、断続的な Puppeteer 起動エラーに対処しました。
- 依存関係のアップグレード – サードパーティ製の依存関係パッケージを一部アップグレードしました。

- Amazon S3 へのアクセス権限のない canary — Amazon S3 へのアクセス権限がなくても canary を実行可能といったバグを修正しました。Amazon S3 へのアクセス権限のないこうした canary では、スクリーンショットなどのアーティファクトを Amazon S3 にアップロードできなくなりました。canary のアクセス権限については、「Canary に必要なロールとアクセス許可」で詳しく確認できます。

> **Important**
>
> 重要: 付属する AWS SDK for JavaScript v2 の依存関係は削除され、将来のランタイムリリースでは AWS SDK for JavaScript v3 を使用するように更新されます。その場合は、canary コードリファレンスを更新します。または、付属する AWS SDK for JavaScript v2 依存関係をソースコードの zip ファイルに依存関係として追加することで、引き続き参照して使用することもできます。

syn-nodejs-puppeteer-6.0

> **Important**
>
> このランタイムバージョンは 2024 年 3 月 8 日に非推奨となる予定です。詳細については、「CloudWatch Synthetics ランタイムサポートポリシー」を参照してください。

主な依存関係:

- Lambda ランタイム Node.js 18.x
- Puppeteer-core バージョン 19.7.0
- Chromium バージョン 111.0.5563.146

syn-nodejs-puppeteer-6.0 の新機能:

- 依存関係のアップグレード – Node.js の依存関係が 18.x にアップグレードされました。
- インターセプトモードのサポート – Puppeteer の協調インターセプトモードのサポートが、Synthetics Canary ランタイムライブラリに追加されました。

- トレーシング動作の変更 – fetch と xhr リクエストのみをトレースし、リソースリクエストはトレースしないようにデフォルトの動作を変更しました。リソースリクエストのトレースは、`traceResourceRequests` オプションを設定することで有効化できます。
- 期間メトリクスの改良 – 今後 `Duration` メトリクスでは、Canary でのアーティファクトのアップロード、スクリーンショットの撮影、CloudWatch メトリクスの生成に要する操作時間が除外されます。 `Duration` メトリクスの値は CloudWatch にレポートされ、Synthetics コンソールで確認することもできます。
- バグ修正: – Canary の実行中に Chromium がクラッシュした場合に生成されるコアダンプをクリーンアップしました。

> **Important**
>
> 重要: 付属する AWS SDK for JavaScript v2 の依存関係は削除され、将来のランタイムリリースでは AWS SDK for JavaScript v3 を使用するように更新されます。その場合は、canary コードリファレンスを更新します。または、付属する AWS SDK for JavaScript v2 依存関係をソースコードの zip ファイルに依存関係として追加することで、引き続き参照して使用することもできます。

syn-nodejs-puppeteer-5.1

> **Important**
>
> このランタイムバージョンは 2024 年 3 月 8 日に非推奨となる予定です。詳細については、「CloudWatch Synthetics ランタイムサポートポリシー」を参照してください。

主な依存関係:

- Lambda ランタイム Node.js 16.x
- Puppeteer-core バージョン 19.7.0
- Chromium バージョン 111.0.5563.146

syn-nodejs-puppeteer-5.1 のバグ修正:

- バグ修正 - このランタイムは、canary によって作成された HAR ファイルにリクエストヘッダーが欠落していた `syn-nodejs-puppeteer-5.0` のバグを修正するものです。

syn-nodejs-puppeteer-5.0

> **Important**
>
> このランタイムバージョンは 2024 年 3 月 8 日に非推奨となる予定です。詳細については、「CloudWatch Synthetics ランタイムサポートポリシー」を参照してください。

主な依存関係:

- Lambda ランタイム Node.js 16.x
- Puppeteer-core バージョン 19.7.0
- Chromium バージョン 111.0.5563.146

syn-nodejs-puppeteer-5.0 の新機能:

- 依存関係のアップグレード — Puppeteer-Core バージョンが 19.7.0 に更新されました。Chromium バージョンは 111.0.5563.146 にアップグレードされました。

> **Important**
>
> 新しい Puppeteer-Core バージョンは、以前のバージョンのPuppeteerと完全に下位互換性があるわけではありません。このバージョンの変更の一部により、廃止された Puppeteer 関数を使用する既存の canary が機能しなくなる可能性があります。詳細については、Puppeteer の変更ログにある Puppeteer Core バージョン 19.7.0 から 6.0 の変更ログの重大な変更点をご覧ください。

syn-nodejs-puppeteer-4.0

> **Important**
>
> このランタイムバージョンは 2024 年 3 月 8 日に非推奨となる予定です。詳細については、「CloudWatch Synthetics ランタイムサポートポリシー」を参照してください。

主な依存関係:

- Lambda ランタイム Node.js 16.x
- Puppeteer-core バージョン 5.5.0
- Chromium バージョン 92.0.4512

syn-nodejs-puppeteer-4.0 の新機能:

- 依存関係のアップグレード — Node.js の依存関係が 16.x にアップデートされました。

Node.js と Puppeteer の廃止されたランタイム

Node.js と Puppeteer では、次のランタイムが廃止されました。

syn-nodejs-puppeteer-3.9

> **Important**
>
> このランタイムバージョンは 2024 年 1 月 8 日に廃止されました。詳細については、「CloudWatch Synthetics ランタイムサポートポリシー」を参照してください。

主な依存関係:

- Lambda ランタイム Node.js 14.x
- Puppeteer-core バージョン 5.5.0
- Chromium バージョン 92.0.4512

syn-nodejs-puppeteer-3.9 の新機能:

- 依存関係のアップグレード – 一部のサードパーティ依存関係パッケージをアップグレードします。

syn-nodejs-puppeteer-3.8

> **Important**
>
> このランタイムバージョンは 2024 年 1 月 8 日に廃止されました。詳細については、「CloudWatch Synthetics ランタイムサポートポリシー」を参照してください。

主な依存関係:

- Lambda ランタイム Node.js 14.x
- Puppeteer-core バージョン 5.5.0
- Chromium バージョン 92.0.4512

syn-nodejs-puppeteer-3.8 の新機能:

- プロファイルのクリーンアップ — Chromium プロファイルが Canary を実行するたびにクリーンアップされるようになりました。

syn-nodejs-puppeteer-3.8 のバグ修正:

- バグ修正 — 以前は、ビジュアルモニターリングの Canary がスクリーンショットがない状態で実行すると誤動作することがありました。この問題が修正されました。

syn-nodejs-puppeteer-3.7

> **Important**
>
> このランタイムバージョンは 2024 年 1 月 8 日に廃止されました。詳細については、「CloudWatch Synthetics ランタイムサポートポリシー」を参照してください。

主な依存関係:

- Lambda ランタイム Node.js 14.x
- Puppeteer-core バージョン 5.5.0
- Chromium バージョン 92.0.4512

syn-nodejs-puppeteer-3.7 の新機能:

- ログ機能の機能強化 — タイムアウトやクラッシュした場合でも、Canary が Amazon S3 にログをアップロードします。
- Lambda レイヤーのサイズを縮小 — Canary に使用される Lambda レイヤーのサイズが 34% 縮小されます。

syn-nodejs-puppeteer-3.7 のバグ修正:

- バグ修正 — 日本語、簡体字中国語、繁体字中国語のフォントが正しくレンダリングされます。

syn-nodejs-puppeteer-3.6

> **Important**
>
> このランタイムバージョンは 2024 年 1 月 8 日に廃止されました。詳細については、「CloudWatch Synthetics ランタイムサポートポリシー」を参照してください。

主な依存関係:

- Lambda ランタイム Node.js 14.x
- Puppeteer-core バージョン 5.5.0
- Chromium バージョン 92.0.4512

syn-nodejs-puppeteer-3.6 の新機能:

- より正確なタイムスタンプ – canary 実行の開始時刻と終了時刻がミリ秒単位まで正確になりました。

syn-nodejs-puppeteer-3.5

> **Important**
>
> このランタイムバージョンは 2024 年 1 月 8 日に廃止されました。詳細については、「CloudWatch Synthetics ランタイムサポートポリシー」を参照してください。

主な依存関係:

- Lambda ランタイム Node.js 14.x
- Puppeteer-core バージョン 5.5.0
- Chromium バージョン 92.0.4512

syn-nodejs-puppeteer-3.5 の新機能:

- 依存関係の更新 - このランタイムの新機能は、更新された依存関係のみです。

syn-nodejs-puppeteer-3.4

> **Important**
>
> このランタイムバージョンは 2022 年 11 月 13 日に廃止されました。詳細については、「CloudWatch Synthetics ランタイムサポートポリシー」を参照してください。

主な依存関係:

- Lambda ランタイム Node.js 12.x
- Puppeteer-core バージョン 5.5.0
- Chromium バージョン 88.0.4298.0

syn-nodejs-puppeteer-3.4 の新機能:

- カスタムハンドラー関数 — Canary スクリプトのためにカスタムハンドラー関数を使用できるようになりました。以前のランタイムでは、スクリプトエントリポイントに `.handler` が含まれている必要がありました。

Canary スクリプトを任意のフォルダに配置し、ハンドラーの一部としてフォルダ名を渡すこともできます。例えば、`MyFolder/MyScriptFile.functionname` はエントリポイントとして使用できます。

- 拡張された HAR ファイル情報 — Canary によって生成された HAR ファイルに、不正なリクエスト、保留中のリクエスト、および不完全なリクエストが表示されるようになりました。

syn-nodejs-puppeteer-3.3

> **Important**
>
> このランタイムバージョンは 2022 年 11 月 13 日に廃止されました。詳細については、「CloudWatch Synthetics ランタイムサポートポリシー」を参照してください。

主な依存関係:

- Lambda ランタイム Node.js 12.x
- Puppeteer-core バージョン 5.5.0
- Chromium バージョン 88.0.4298.0

syn-nodejs-puppeteer-3.3 の新機能:

- アーティファクト暗号化のその他のオプション– Canary では、Amazon S3 に保存したアーティファクトの暗号化に AWS 管理キーを使用する代わりに、このバージョン以降のランタイムを使用します。使用するキーは、AWS KMS カスタマー管理のキーまたは Amazon S3 が管理するキーから選択できます。詳細については、「Canary アーティファクトの暗号化」を参照してください。

syn-nodejs-puppeteer-3.2

> **Important**
>
> このランタイムバージョンは 2022 年 11 月 13 日に廃止されました。詳細については、「CloudWatch Synthetics ランタイムサポートポリシー」を参照してください。

主な依存関係:

- Lambda ランタイム Node.js 12.x
- Puppeteer-core バージョン 5.5.0
- Chromium バージョン 88.0.4298.0

syn-nodejs-puppeteer-3.2 の新機能:

- スクリーンショットによるビジュアルモニターリング— これ以降のランタイムを使用する Canary は、実行中に撮影されたスクリーンショットと、同じスクリーンショットのベースラインバージョンを比較できます。スクリーンショットが指定されたパーセンテージのしきい値よりも大きく異なる場合、Canary は失敗します。詳細については、「ビジュアルモニターリング」または「ビジュアルモニターリングの設計図」を参照してください。
- 機密データに関する新機能 Canary ログやレポートに機密データが表示されないようにすることができます。詳細については、「SyntheticsLogHelper クラス」を参照してください。
- 廃止される関数 `RequestResponseLogHelper`クラスは廃止され、他の新しい設定オプションに置き換えられます。詳細については、「RequestResponseLogHelper クラス」を参照してください。

syn-nodejs-puppeteer-3.1

> **Important**
>
> このランタイムバージョンは 2022 年 11 月 13 日に廃止されました。詳細については、「CloudWatch Synthetics ランタイムサポートポリシー」を参照してください。

主な依存関係:

- Lambda ランタイム Node.js 12.x
- Puppeteer-core バージョン 5.5.0
- Chromium バージョン 88.0.4298.0

syn-nodejs-puppeteer-3.1 の新機能:

- CloudWatch メトリクスの設定機能 – このランタイムでは、不要なメトリクスを無効にできます。それ以外の場合、Canary は Canary 実行ごとにさまざまな CloudWatch メトリクスを発行します。

- スクリーンショットリンク – ステップの完了後に、スクリーンショットを Canary ステップにリンクできます。これを行うには、スクリーンショットを関連付けるステップの名前を使用して、takeScreenshot メソッドでスクリーンショットを作成します。例えば、ステップを実行し、待機時間を追加して、スクリーンショットを作成できます。
- ハートビートモニターリンググループリントは複数の URL のモニターリングが可能 – CloudWatch コンソールでハートビートモニターリンググループリントを使用して複数の URL をモニターリングし、Canary 実行レポートのステップの概要で各 URL のステータス、期間、関連付けられたスクリーンショット、失敗の理由を確認できます。

syn-nodejs-puppeteer-3.0

> **Important**
>
> このランタイムバージョンは 2022 年 11 月 13 日に廃止されました。詳細については、「CloudWatch Synthetics ランタイムサポートポリシー」を参照してください。

主な依存関係:

- Lambda ランタイム Node.js 12.x
- Puppeteer-core バージョン 5.5.0
- Chromium バージョン 88.0.4298.0

syn-nodejs-puppeteer-3.0 の新機能:

- アップグレードされた依存関係 – このバージョンは Puppeteer バージョン 5.5.0、Node.js 12.x、および Chromium 88.0.4298.0 を使用します。
- クロスリージョンバケットアクセス – Canary がログファイル、スクリーンショット、HAR ファイルを保存するバケットとして、別のリージョンの S3 バケットを指定できるようになりました。
- 新しい関数が利用可能 – このバージョンでは、Canary 名と Syntheticsのランタイムバージョンを取得するためのライブラリ関数を追加します。

  詳細については、「Synthetics クラス」を参照してください。

syn-nodejs-2.2

このセクションでは、`syn-nodejs-2.2` ランタイムバージョンについて説明します。

> **Important**
>
> このランタイムバージョンは 2021 年 5 月 28 日に廃止されました。詳細については、「CloudWatch Synthetics ランタイムサポートポリシー」を参照してください。

主な依存関係:

- Lambda ランタイム Node.js 10.x
- Puppeteer-core バージョン 3.3.0
- Chromium バージョン 83.0.4103.0

syn-nodejs-2.2 の新機能:

- HTTP ステップとして Canary をモニターリングする – 単一の Canary で複数の API をテストできるようになりました。各 API は個別の HTTP ステップとしてテストされ、CloudWatch Synthetics はステップメトリクスと CloudWatch Synthetics のステップレポートを使用して各ステップのステータスをモニターリングします。CloudWatch Synthetics では、HTTP ステップごとに `SuccessPercent` および `Duration` メトリクスが作成されます。

  この機能は、executeHttpStep(stepName, requestOptions, callback, stepConfig) 関数によって実装されます。詳細については、「executeHttpStep(stepName, requestOptions, [callback], [stepConfig])」を参照してください。

  API Canary ブループリントは、この新しい機能を使用するように更新されました。
- HTTP リクエストレポート – リクエスト/レスポンスヘッダー、レスポンス本文、ステータスコード、エラーとパフォーマンスのタイミング、TCP 接続時間、TLS ハンドシェイク時間、最初のバイト時間、コンテンツ転送時間などの詳細を取得する HTTP リクエストレポートを表示できるようになりました。HTTP/HTTPS モジュールを内部で使用するすべての HTTP リクエストは、ここで取得されます。ヘッダーとレスポンス本文はデフォルトでは取得されませんが、設定オプションを設定することで有効にできます。
- グローバルおよびステップレベルの設定 – CloudWatch Synthetics 設定は、Canary のすべてのステップに適用されるグローバルレベルで設定できます。特定のオプションを有効または無効にするために、設定キーと値のペアを渡して、これらの設定をステップレベルで上書きすることもできます。

  詳細については、「SyntheticsConfiguration クラス」を参照してください。

- ステップ失敗時に続行する設定 – ステップが失敗した場合に、Canary の実行を続行できます。`executeHttpStep` 関数では、これはデフォルトでオンになっています。このオプションは、グローバルレベルで一度設定することも、ステップごとに個別に設定することもできます。

syn-nodejs-2.1

> **Important**
>
> このランタイムバージョンは 2021 年 5 月 28 日に廃止されました。詳細については、「CloudWatch Synthetics ランタイムサポートポリシー」を参照してください。

主な依存関係:

- Lambda ランタイム Node.js 10.x
- Puppeteer-core バージョン 3.3.0
- Chromium バージョン 83.0.4103.0

syn-nodejs-2.1 の新機能:

- 設定可能なスクリーンショットの動作 – UI Canary によるスクリーンショットのキャプチャをオフにする機能を提供します。以前のバージョンのランタイムを使用する Canary の場合、UI Canary は常に各ステップの前後にスクリーンショットをキャプチャします。`syn-nodejs-2.1` の場合、これは設定可能です。スクリーンショットをオフにすると、Amazon S3 のストレージコストが削減され、HIPAA 規制に準拠しやすくなります。詳細については、「SyntheticsConfiguration クラス」を参照してください。
- Google Chrome 起動パラメータのカスタマイズ Canary が Google Chrome ブラウザウィンドウを起動するときに使用する引数を設定できるようになりました。詳細については、「launch(options)」を参照してください。

以前のバージョンの Canary ランタイムと比べて、syn-nodejs-2.0 以降を使用した場合の Canary の実行時間は少し増えることがあります。

syn-nodejs-2.0

> **Important**
>
> このランタイムバージョンは 2021 年 5 月 28 日に廃止されました。詳細については、「CloudWatch Synthetics ランタイムサポートポリシー」を参照してください。

主な依存関係:

- Lambda ランタイム Node.js 10.x
- Puppeteer-core バージョン 3.3.0
- Chromium バージョン 83.0.4103.0

syn-nodejs-2.0 の新機能:

- アップグレードされた依存関係 – このランタイムバージョンでは、Puppeteer-core バージョン 3.3.0 と Chromium バージョン 83.0.4103.0 を使用します
- X-RAY アクティブトレースのサポート。Canary でトレースが有効になっている場合、ブラウザ、AWS SDK、または HTTP または HTTPS モジュールを使用するCanary によって行われたすべての呼び出しに対して Xx-Ray トレースが送信されます。トレースが有効になっている canary は、トレースが有効になっている他のサービスやアプリケーションにリクエストを送信していない場合でも、X-Ray トレースマップに表示されます。詳細については、「Canary と X-Ray のトレース」を参照してください。
- Synthetics レポート – Canary 実行ごとに、`SyntheticsReport-PASSED.json` または `SyntheticsReport-FAILED.json` という名前のレポートが CloudWatch Synthetics により作成され、開始時刻、終了時刻、ステータス、エラーなどのデータが記録されます。また、Canary スクリプトの各ステップの PASSED/FAILED ステータス、および各ステップでキャプチャされたエラーとスクリーンショットも記録されます。
- リンク切れチェッカーレポート – このランタイムに含まれるリンク切れチェッカーの新しいバージョンでは、チェックされたリンク、ステータスコード、エラーの理由 (ある場合)、およびソースページとターゲットページのスクリーンショットを含むレポートが作成されます。
- 新しい CloudWatch メトリクス – Synthetics は、`2xx`、`4xx`、`5xx`、および `RequestFailed` という名前のメトリクスを `CloudWatchSynthetics` 名前空間で公開します。これらのメトリクスには、Canary 実行の 200 番台、400 番台、500 番台、およびリクエストのエラー数が示されます。このランタイムバージョンでは、これらのメトリクスは UI Canary に対してのみレポート

され、API Canary に対してはレポートされません。また、ランタイムバージョン `syn-nodejs-puppeteeer-2.2` で開始される API Canary についても報告されます。

- ソート可能な HAR ファイル – ステータスコード、リクエストサイズ、期間によって HAR ファイルをソートできるようになりました。
- メトリクスのタイムスタンプ – CloudWatch メトリクスは、Canary 実行終了時刻ではなく、Lambda 呼び出し時刻に基づいて報告されるようになりました。

syn-nodejs-2.0 のバグ修正:

- Canary アーティファクトアップロードエラーが報告されない問題を修正しました。このようなエラーは、実行エラーとして表示されます。
- リダイレクトされたリクエスト (3xx) が誤ってエラーとして記録される問題を修正しました。
- スクリーンショットの番号が 0 から始まる問題を修正しました。今後は 1 から始まります。
- 中国語と日本語のフォントでスクリーンショットが文字化けする問題を修正しました。

以前のバージョンの Canary ランタイムと比べて、syn-nodejs-2.0 以降を使用した場合の Canary の実行時間は少し増えることがあります。

syn-nodejs-2.0-beta

> **Important**
>
> このランタイムバージョンは 2021 年 2 月 8 日に非推奨となりました。詳細については、「CloudWatch Synthetics ランタイムサポートポリシー」を参照してください。

主な依存関係:

- Lambda ランタイム Node.js 10.x
- Puppeteer-core バージョン 3.3.0
- Chromium バージョン 83.0.4103.0

syn-nodejs-2.0-beta の新機能:

- アップグレードされた依存関係 – このランタイムバージョンでは、Puppeteer-core バージョン 3.3.0 と Chromium バージョン 83.0.4103.0 を使用します

- Synthetics レポート – Canary 実行ごとに、`SyntheticsReport-PASSED.json` または `SyntheticsReport-FAILED.json` という名前のレポートが CloudWatch Synthetics により作成され、開始時刻、終了時刻、ステータス、エラーなどのデータが記録されます。また、Canary スクリプトの各ステップの PASSED/FAILED ステータス、および各ステップでキャプチャされたエラーとスクリーンショットも記録されます。
- リンク切れチェッカーレポート – このランタイムに含まれるリンク切れチェッカーの新しいバージョンでは、チェックされたリンク、ステータスコード、エラーの理由 (ある場合)、およびソースページとターゲットページのスクリーンショットを含むレポートが作成されます。
- 新しい CloudWatch メトリクス – Synthetics は、`2xx`、`4xx`、`5xx`、および `RequestFailed` という名前のメトリクスを `CloudWatchSynthetics` 名前空間で公開します。これらのメトリクスには、Canary 実行の 200 番台、400 番台、500 番台、およびリクエストのエラー数が示されます。これらのメトリクスは UI Canary に対してのみレポートされ、API Canary に対してはレポートされません。
- ソート可能な HAR ファイル – ステータスコード、リクエストサイズ、期間によって HAR ファイルをソートできるようになりました。
- メトリクスのタイムスタンプ – CloudWatch メトリクスは、Canary 実行終了時刻ではなく、Lambda 呼び出し時刻に基づいて報告されるようになりました。

syn-nodejs-2.0-beta のバグ修正:

- Canary アーティファクトアップロードエラーが報告されない問題を修正しました。このようなエラーは、実行エラーとして表示されます。
- リダイレクトされたリクエスト (3xx) が誤ってエラーとして記録される問題を修正しました。
- スクリーンショットの番号が 0 から始まる問題を修正しました。今後は 1 から始まります。
- 中国語と日本語のフォントでスクリーンショットが文字化けする問題を修正しました。

syn-1.0

> **Important**
>
> このランタイムバージョンは 2021 年 5 月 28 日に非推奨となる予定です。詳細については、「CloudWatch Synthetics ランタイムサポートポリシー」を参照してください。

最初の Synthetics ランタイムバージョンは `syn-1.0` です。

主な依存関係:

- Lambda ランタイム Node.js 10.x
- Puppeteer-core バージョン 1.14.0
- Puppeteer-core 1.14.0 と一致する Chromium バージョン

Python と Selenium Webdriver を使用するランタイムバージョン

次のセクションには、Python と Selenium Webdriver のための CloudWatch Synthetics ランタイムバージョンに関する情報が含まれています。Selenium は、オープンソースのブラウザ自動化ツールです。Selenium の詳細については、www.selenium.dev/ をご参照ください。

これらのランタイムバージョンの命名規則は `syn-language-framework-majorversion.minorversion` です。

> **Important**
>
> 次の CloudWatch Synthetics ランタイムバージョンは 2024 年 3 月 8 日に廃止される予定です。
>
> - `syn-python-selenium-2.0`
> - `syn-python-selenium-1.3`
> - `syn-python-selenium-1.2`
> - `syn-python-selenium-1.1`
> - `syn-python-selenium-1.0`
>
> 詳細については、「CloudWatch Synthetics ランタイムサポートポリシー」を参照してください。

syn-python-selenium-3.0

バージョン 3.0 は Python および Selenium 用の最新の CloudWatch Synthetics ランタイムです。

主な依存関係:

- Python 3.8

- Selenium 4.15.1
- Chromium バージョン 121.0.6167.139

syn-python-selenium-3.0 の新機能:

- Chromium でバンドルされたライブラリのバージョンを更新 — Chromium の依存関係が新しいバージョンに更新されました。

syn-python-selenium-2.1

主な依存関係:

- Python 3.8
- Selenium 4.15.1
- Chromium バージョン 111.0.5563.146

syn-python-selenium-2.1 の新機能:

- Chromium でバンドルされたライブラリのバージョンを更新 — Chromium と Selenium の依存関係が新しいバージョンに更新されました。

syn-python-selenium-2.0

> **Important**
>
> このランタイムバージョンは 2024 年 3 月 8 日に非推奨となる予定です。詳細については、「CloudWatch Synthetics ランタイムサポートポリシー」を参照してください。

主な依存関係:

- Python 3.8
- Selenium 4.10.0
- Chromium バージョン 111.0.5563.146

syn-python-selenium-2.0 の新機能:

- 依存関係を更新 — Chromium と Selenium の依存関係が新しいバージョンに更新されました。

syn-python-selenium-2.0 のバグ修正:

- タイムスタンプの追加 — Canary ログにタイムスタンプが追加されました。
- セッションの再利用 — バグが修正され、Canary が前回実行したセッションを再利用できなくなりました。

syn-python-selenium-1.3

> **Important**
>
> このランタイムバージョンは 2024 年 3 月 8 日に非推奨となる予定です。詳細については、「CloudWatch Synthetics ランタイムサポートポリシー」を参照してください。

主な依存関係:

- Python 3.8
- Selenium 3.141.0
- Chromium バージョン 92.0.4512.0

syn-python-selenium-1.3 の新機能:

- より正確なタイムスタンプ – canary 実行の開始時刻と終了時刻がミリ秒単位まで正確になりました。

syn-python-selenium-1.2

> **Important**
>
> このランタイムバージョンは 2024 年 3 月 8 日に非推奨となる予定です。詳細については、「CloudWatch Synthetics ランタイムサポートポリシー」を参照してください。

主な依存関係:

- Python 3.8
- Selenium 3.141.0
- Chromium バージョン 92.0.4512.0

- 依存関係の更新 - このランタイムの新機能は、更新された依存関係のみです。

syn-python-selenium-1.1

> **Important**
>
> このランタイムバージョンは 2024 年 3 月 8 日に非推奨となる予定です。詳細については、「CloudWatch Synthetics ランタイムサポートポリシー」を参照してください。

主な依存関係:

- Python 3.8
- Selenium 3.141.0
- Chromium バージョン 83.0.4103.0

機能:

- カスタムハンドラー関数 — Canary スクリプトのためにカスタムハンドラー関数を使用できるようになりました。以前のランタイムでは、スクリプトエントリポイントに `.handler` が含まれている必要がありました。

  Canary スクリプトを任意のフォルダに配置し、ハンドラーの一部としてフォルダ名を渡すこともできます。例えば、`MyFolder/MyScriptFile.functionname` はエントリポイントとして使用できます。
- メトリクスとステップ障害の設定を追加するための設定オプション — これらのオプションは、Node.js Canary のランタイムで既に使用可能となっていました。詳細については、「SyntheticsConfiguration クラス」を参照してください。
- Chrome のカスタム引数 — ブラウザをシークレットモードで開くか、プロキシサーバー設定を渡すことができるようになりました。詳細については、「 Chrome()」を参照してください。
- クロスリージョンアーティファクトバケット - Canary は、アーティファクトを別のリージョンの Amazon S3 バケットに保存できます。

- **`index.py` の問題の修正を含むバグ修正** — 以前のランタイムでは、`index.py` という名前の付いた Canary ファイルがライブラリファイルの名前と競合していたため、例外が発生していました。この問題は修正されました。

syn-python-selenium-1.0

> **Important**
>
> このランタイムバージョンは 2024 年 3 月 8 日に非推奨となる予定です。詳細については、「CloudWatch Synthetics ランタイムサポートポリシー」を参照してください。

主な依存関係:

- Python 3.8
- Selenium 3.141.0
- Chromium バージョン 83.0.4103.0

機能:

- Selenium サポート – Selenium テストフレームワークを使用して Canary スクリプトを記述できます。Selenium スクリプトを他の場所から CloudWatch Synthetics に最小限の変更で持ち込むことができ、AWS のサービスで動作します。

Canary スクリプトの作成

次のセクションでは、Canary スクリプトの作成方法と、Canary を他の AWS のサービス、外部依存関係、ライブラリと統合する方法について説明します。

トピック

- Node.js Canary スクリプトの記述
- Python Canary スクリプトの記述
- Synthetics Canary を使用するようにするための既存の Selenium の変更
- 既存の Puppeteer Synthetics スクリプトを変更して非標準の証明書を認証する

Node.js Canary スクリプトの記述

トピック

CloudWatch Synthetics Canary を最初から作成する

次の例は、Synthetics の最小の Canary スクリプトを示しています。このスクリプトは、合格して正常な実行となり、文字列を返します。不合格となる Canary の例を確認するには、`let fail = false;` を `let fail = true;` に変更します。

Canary スクリプトのエントリポイント関数を定義する必要があります。Canary の `ArtifactS3Location` として指定した先の Amazon S3 にファイルがアップロードされる方法を確認するには、これらのファイルを /tmp フォルダの下に作成します。スクリプトを実行すると、合格/不合格のステータスと所要時間のメトリクスが CloudWatch に発行され、/tmp の下のファイルが S3 にアップロードされます。

```
const basicCustomEntryPoint = async function () {

    // Insert your code here

    // Perform multi-step pass/fail check

    // Log decisions made and results to /tmp

    // Be sure to wait for all your code paths to complete
    // before returning control back to Synthetics.
    // In that way, your canary will not finish and report success
    // before your code has finished executing

    // Throw to fail, return to succeed
    let fail = false;
    if (fail) {
        throw "Failed basicCanary check.";
```

```
    }

    return "Successfully completed basicCanary checks.";
};

exports.handler = async () => {
    return await basicCustomEntryPoint();
};
```

次に、AWS SDK を使用して、Synthetics のログ記録を使用して呼び出しを行うようにスクリプトが拡張されます。デモのために、このスクリプトは Amazon DynamoDB クライアントを作成し、DynamoDB listTables API を呼び出します。リクエストに対するレスポンスを記録し、リクエストが成功したかどうかに応じて合格または不合格を記録します。

```
const log = require('SyntheticsLogger');
const AWS = require('aws-sdk');
// Require any dependencies that your script needs
// Bundle additional files and dependencies into a .zip file with folder structure
// nodejs/node_modules/additional files and folders

const basicCustomEntryPoint = async function () {

    log.info("Starting DynamoDB:listTables canary.");

    let dynamodb = new AWS.DynamoDB();
    var params = {};
    let request = await dynamodb.listTables(params);
    try {
        let response = await request.promise();
        log.info("listTables response: " + JSON.stringify(response));
    } catch (err) {
        log.error("listTables error: " + JSON.stringify(err), err.stack);
        throw err;
    }

    return "Successfully completed DynamoDB:listTables canary.";
};

exports.handler = async () => {
    return await basicCustomEntryPoint();
};
```

Node.js Canary ファイルのパッケージング

Amazon S3 の場所を使用して Canary スクリプトをアップロードする場合、zip ファイルにはフォルダ構造 nodejs/node_modules/*myCanaryFilename.js file* にスクリプトが含まれている必要があります。

複数の .js ファイルがある場合や、スクリプトに依存関係がある場合は、それらのすべてを単一の ZIP ファイル (フォルダ構造は nodejs/node_modules/*myCanaryFilename.js file and other folders and files*) にバンドルできます。syn-nodejs-puppeteer-3.4 以降を使用している場合は、オプションで Canary ファイルを別のフォルダーに配置し、次のようなフォルダー構造を作成できます: nodejs/node_modules/*myFolder*/*myCanaryFilename.js file and other folders and files*。

ハンドラー名

スクリプトのエントリポイント (ハンドラー) のファイル名に一致するように、Canary のスクリプトのエントリポイントを myCanaryFilename.functionName として設定します。syn-nodejs-puppeteer-3.4 より前のランタイムを使用している場合は、functionName は handler である必要があります。syn-nodejs-puppeteer-3.4 以降を使用している場合、ハンドラーとして任意の関数名を選択できます。syn-nodejs-puppeteer-3.4 以降を使用している場合、オプションで Canary を nodejs/node_modules/myFolder/my_canary_filename などの別のフォルダに保存することもできます。別のフォルダに保存する場合は、スクリプトエントリポイントでそのパスを指定します (myFolder/my_canary_filename.functionName など)。

既存の Puppeteer スクリプトを変更して Synthetics の Canary として使用する

このセクションでは、Puppeteer スクリプトを変更して Synthetics の Canary スクリプトとして実行する方法について説明します。Puppeteer の詳細については、「Puppeteer API v1.14.0」を参照してください。

まず、次の Puppeteer スクリプトの例から始めます。

```
const puppeteer = require('puppeteer');

(async () => {
  const browser = await puppeteer.launch();
  const page = await browser.newPage();
  await page.goto('https://example.com');
  await page.screenshot({path: 'example.png'});
```

```
  await browser.close();
})();
```

変更の手順は次のとおりです。

- handler 関数を作成してエクスポートします。このハンドラーは、スクリプトのエントリポイント関数です。syn-nodejs-puppeteer-3.4 より前のランタイムを使用している場合、ハンドラー関数には handler という名前を付ける必要があります。syn-nodejs-puppeteer-3.4 以降を使用している場合、関数には任意の名前を付けることができますが、スクリプトで使用されている名前と同じである必要があります。また、syn-nodejs-puppeteer-3.4 以降を使用している場合は、スクリプトを任意のフォルダに保存し、そのフォルダをハンドラー名の一部として指定できます。

  ```
  const basicPuppeteerExample = async function () {};

  exports.handler = async () => {
      return await basicPuppeteerExample();
  };
  ```

- Synthetics 依存関係を使用します。

  ```
  var synthetics = require('Synthetics');
  ```

- Puppeteer の Page オブジェクトを取得するには、Synthetics.getPage 関数を使用します。

  ```
  const page = await synthetics.getPage();
  ```

  Synthetics.getPage 関数から返されるページオブジェクトには、ログ記録用にインストルメント化された page.on、request、response、および requestfailed の各イベントがあります。また、Synthetics は、ページのリクエストおよびレスポンス用の HAR ファイルの生成を設定し、ページの送信リクエストのユーザーエージェントヘッダーに Canary ARN を追加します。

これで、スクリプトを Synthetics の Canary として実行できるようになりました。更新されたスクリプトは次のとおりです。

```
var synthetics = require('Synthetics');  // Synthetics dependency

const basicPuppeteerExample = async function () {
    const page = await synthetics.getPage(); // Get instrumented page from Synthetics
```

```
    await page.goto('https://example.com');
    await page.screenshot({path: '/tmp/example.png'}); // Write screenshot to /tmp
 folder
};

exports.handler = async () => {  // Exported handler function
    return await basicPuppeteerExample();
};
```

環境変数

Canary を作成する際に環境変数を使用できます。これにより、単一の Canary スクリプトを記述し、そのスクリプトを異なる値で使用して、同様のタスクを持つ複数の Canary をすばやく作成できます。

例えば、組織が、ソフトウェア開発のさまざまな段階向けに prod、dev、pre-release などのエンドポイントを有しており、これらの各エンドポイントをテストするために Canary を作成する必要があるとします。ソフトウェアをテストする 1 つの Canary スクリプトを記述し、3 つの Canary をそれぞれ作成するときに、エンドポイント環境変数に異なる値を指定できます。その後、Canary を作成するときに、環境変数に使用するスクリプトと値を指定します。

環境変数の名前には、文字、数字、およびアンダースコアを使用できます。文字で始まり、少なくとも 2 文字である必要があります。環境変数の合計サイズは 4 KB を超えることはできません。Lambda の予約済み環境変数を環境変数の名前として指定することはできません。予約済み環境変数の詳細については、「ランタイム環境変数」をご参照ください。

> **Important**
>
> 環境変数のキーと値は暗号化されません。機密情報は保存しないでください。

次のスクリプト例では、2 つの環境変数を使用しています。このスクリプトは、ウェブページが利用可能かどうかをチェックする Canary 用です。環境変数を使用して、チェックする URL と、使用する CloudWatch Synthetics ログレベルの両方をパラメータ化します。

次の関数は、LogLevel を LOG_LEVEL 環境変数の値に設定します。

```
synthetics.setLogLevel(process.env.LOG_LEVEL);
```

この関数は、URL を URL 環境変数の値に設定します。

```
const URL = process.env.URL;
```

これは完全なスクリプトです。このスクリプトを使用して Canary を作成するときは、LOG_LEVEL および URL 環境変数の値を指定します。

```
var synthetics = require('Synthetics');
const log = require('SyntheticsLogger');

const pageLoadEnvironmentVariable = async function () {

    // Setting the log level (0-3)
    synthetics.setLogLevel(process.env.LOG_LEVEL);
    // INSERT URL here
    const URL = process.env.URL;

    let page = await synthetics.getPage();
    //You can customize the wait condition here. For instance,
    //using 'networkidle2' may be less restrictive.
    const response = await page.goto(URL, {waitUntil: 'domcontentloaded', timeout:
 30000});
    if (!response) {
        throw "Failed to load page!";
    }
    //Wait for page to render.
    //Increase or decrease wait time based on endpoint being monitored.
    await page.waitFor(15000);
    await synthetics.takeScreenshot('loaded', 'loaded');
    let pageTitle = await page.title();
    log.info('Page title: ' + pageTitle);
    log.debug('Environment variable:' + process.env.URL);

    //If the response status code is not a 2xx success code
    if (response.status() < 200 || response.status() > 299) {
        throw "Failed to load page!";
    }
};

exports.handler = async () => {
    return await pageLoadEnvironmentVariable();
};
```

環境変数をスクリプトに渡す

コンソールで Canary を作成するときに環境変数をスクリプトに渡すには、コンソールの [Environment variables] (環境変数) セクションで環境変数のキーと値を指定します。詳細については、「Canary を作成する」を参照してください。

API または AWS CLI を介して環境変数を渡すには、RunConfig セクションの EnvironmentVariables パラメータを使用します。以下は、キー Environment とキー Region を持つ 2 つの環境変数を使用する Canary を作成する AWS CLI コマンドの例です。

```
aws synthetics create-canary --cli-input-json '{
   "Name":"nameofCanary",
   "ExecutionRoleArn":"roleArn",
   "ArtifactS3Location":"s3://cw-syn-results-123456789012-us-west-2",
   "Schedule":{
      "Expression":"rate(0 minute)",
      "DurationInSeconds":604800
   },
   "Code":{
      "S3Bucket": "canarycreation",
      "S3Key": "cwsyn-mycanaryheartbeat-12345678-d1bd-1234-
abcd-123456789012-12345678-6a1f-47c3-b291-123456789012.zip",
      "Handler":"pageLoadBlueprint.handler"
   },
   "RunConfig": {
      "TimeoutInSeconds":60,
      "EnvironmentVariables": {
         "Environment":"Production",
         "Region": "us-west-1"
      }
   },
   "SuccessRetentionPeriodInDays":13,
   "FailureRetentionPeriodInDays":13,
   "RuntimeVersion":"syn-nodejs-2.0"
}'
```

Canary と他の AWS のサービスとの統合

すべての Canary では AWS SDK ライブラリを使用できます。このライブラリを Canary の作成時に使用すると、Canary を他の AWS のサービスと統合できます。

これを行うには、Canary に次のコードを追加する必要があります。これらの例では、Canary に統合されているサービスとして AWS Secrets Manager が使用されます。

- AWS SDK をインポートします。

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

- 統合する AWS のサービスのクライアントを作成します。

```
const secretsManager = new AWS.SecretsManager();
```

- このクライアントを使用して、サービスへの API コールを行います。

```
var params = {
  SecretId: secretName
};
return await secretsManager.getSecretValue(params).promise();
```

次の Canary スクリプトのコードスニペットは、Secrets Manager との統合例をより詳細に示しています。

```
var synthetics = require('Synthetics');
const log = require('SyntheticsLogger');

const AWS = require('aws-sdk');
const secretsManager = new AWS.SecretsManager();

const getSecrets = async (secretName) => {
    var params = {
        SecretId: secretName
    };
    return await secretsManager.getSecretValue(params).promise();
}

const secretsExample = async function () {
    let URL = "<URL>";
    let page = await synthetics.getPage();

    log.info(`Navigating to URL: ${URL}`);
    const response = await page.goto(URL, {waitUntil: 'domcontentloaded', timeout:
 30000});

    // Fetch secrets
    let secrets = await getSecrets("secretname")
```

```
    /**
    * Use secrets to login.
    *
    * Assuming secrets are stored in a JSON format like:
    * {
    *   "username": "<USERNAME>",
    *   "password": "<PASSWORD>"
    * }
    **/
    let secretsObj = JSON.parse(secrets.SecretString);
    await synthetics.executeStep('login', async function () {
        await page.type(">USERNAME-INPUT-SELECTOR<", secretsObj.username);
        await page.type(">PASSWORD-INPUT-SELECTOR<", secretsObj.password);

        await Promise.all([
          page.waitForNavigation({ timeout: 30000 }),
          await page.click(">SUBMIT-BUTTON-SELECTOR<")
        ]);
    });

    // Verify login was successful
    await synthetics.executeStep('verify', async function () {
        await page.waitForXPath(">SELECTOR<", { timeout: 30000 });
    });
};

exports.handler = async () => {
    return await secretsExample();
};
```

Canary に静的 IP アドレスの使用を強制する

静的 IP アドレスを使用するように Canary を設定できます。

Canary に静的 IP アドレスの使用を強制するには

1. 新しい VPC を作成します。詳細については、「VPC での DNS の使用」を参照してください。
2. 新しいインターネットゲートウェイを作成します。詳細については、「インターネットゲートウェイを VPC に追加する」を参照してください。
3. 新しい VPC 内にパブリックサブネットを作成します。
4. 新しいルートテーブルを VPC に追加します。

5. `0.0.0.0/0` からインターネットゲートウェイに向かうルートを、新しいルートテーブルに追加します。
6. 新しいルートテーブルをパブリックサブネットに関連付けます。
7. Elastic IP アドレスを作成します。詳細については、「Elastic IP アドレス」を参照してください。
8. 新しい NAT ゲートウェイを作成し、パブリックサブネットと Elastic IP アドレスに割り当てます。
9. VPC の内部にプライベートサブネットを作成します。
10. `0.0.0.0/0` から NAT ゲートウェイへのルートを VPC デフォルトルートテーブルに追加する
11. Canary を作成します。

Python Canary スクリプトの記述

このスクリプトは、合格して正常な実行となり、文字列を返します。失敗した Canary がどのように見えるかを確認するには、fail = False を fail = True に変更します

```
def basic_custom_script():
    # Insert your code here
    # Perform multi-step pass/fail check
    # Log decisions made and results to /tmp
    # Be sure to wait for all your code paths to complete
    # before returning control back to Synthetics.
    # In that way, your canary will not finish and report success
    # before your code has finished executing
    fail = False
    if fail:
        raise Exception("Failed basicCanary check.")
    return "Successfully completed basicCanary checks."
def handler(event, context):
    return basic_custom_script()
```

Python Canary ファイルのパッケージング

複数の .py ファイルがある場合、またはスクリプトに依存関係がある場合は、それらすべてを単一の ZIP ファイルにバンドルできます。`syn-python-selenium-1.1` ランタイムを使用する場合、ZIP ファイルには、`python/my_canary_filename.py` などの `python` フォルダ内にメインの Canary .py ファイルが含まれている必要があります。`syn-python-selenium-1.1` 以降を使用

する場合は、オプションで、python/myFolder/my_canary_filename.py などの別のフォルダを使用できます。

この ZIP ファイルには、必要なすべてのフォルダとファイルが含まれている必要がありますが、他のファイルは python フォルダ内にある必要はありません。

スクリプトのエントリポイントのファイル名および関数名に一致するように、Canary のスクリプトのエントリポイントを my_canary_filename.functionName として設定します。syn-python-selenium-1.0 ランタイムを使用している場合は、functionName は handler である必要があります。syn-python-selenium-1.1 以降を使用している場合、このハンドラー名の制限は適用されません。また、オプションで Canary を python/myFolder/my_canary_filename.py などの別のフォルダに保存することもできます。別のフォルダに保存する場合は、スクリプトエントリポイントでそのパスを指定します (myFolder/my_canary_filename.functionName など)。

Synthetics Canary を使用するようにするための既存の Selenium の変更

Canary として使用するために、Python と Selenium の既存のスクリプトをすばやく変更できます。Selenium の詳細については、www.selenium.dev/ をご参照ください。

この例では、次の Selenium スクリプトから始めます。

```
from selenium import webdriver

def basic_selenium_script():
    browser = webdriver.Chrome()
    browser.get('https://example.com')
    browser.save_screenshot('loaded.png')

basic_selenium_script()
```

変更の手順は次のとおりです。

Selenium スクリプトを Canary として使用するように変換するには

1. aws_synthetics モジュールから Selenium を使用するように import ステートメントを変更します。

   ```
   from aws_synthetics.selenium import synthetics_webdriver as webdriver
   ```

   aws_synthetics の Selenium モジュールは、Canary がメトリクスとログを出力し、HAR ファイルを生成し、他の CloudWatch Synthetics 機能で動作することを保証します。

2. ハンドラ関数を作成し、Selenium メソッドを呼び出します。このハンドラーは、スクリプトのエントリポイント関数です。

   syn-python-selenium-1.0 を使用している場合、ハンドラー関数には handler という名前を付ける必要があります。syn-python-selenium-1.1 以降を使用している場合、関数には任意の名前を付けることができますが、スクリプトで使用されている名前と同じである必要があります。また、syn-python-selenium-1.1 以降を使用している場合は、スクリプトを任意のフォルダに保存し、そのフォルダをハンドラー名の一部として指定できます。

   ```
   def handler(event, context):
       basic_selenium_script()
   ```

これで、スクリプトが CloudWatch Synthetics Canary に更新されました。更新されたスクリプトは次のとおりです。

```
from aws_synthetics.selenium import synthetics_webdriver as webdriver

def basic_selenium_script():
    browser = webdriver.Chrome()
    browser.get('https://example.com')
    browser.save_screenshot('loaded.png')

def handler(event, context):
    basic_selenium_script()
```

既存の Puppeteer Synthetics スクリプトを変更して非標準の証明書を認証する

Synthetics Canaries の重要なユースケースの 1 つは、独自のエンドポイントをモニタリングすることです。外部トラフィックに対応していないエンドポイントをモニタリングする場合、信頼できるサードパーティーの認証局によって署名された適切な証明書が存在しない可能性があります。

このシナリオで考えられる解決策は、次の 2 つです。

- クライアント証明書を認証するには、「How to validate authentication using Amazon CloudWatch Synthetics – Part 2」を参照してください。
- 自己署名証明書を認証するには、「How to validate authentication with self-signed certificates in Amazon CloudWatch Synthetics」を参照してください。

CloudWatch Synthetics Canary を使用する場合は、これら 2 つのオプションに限定されません。Canary コードを拡張することで、これらの機能を拡張し、ビジネスロジックを追加できます。

> **Note**
>
> Python ランタイムで実行される Synthetics Canary は、もともと `--ignore-certificate-errors` フラグが有効になっているため、これらの Canary が非標準の証明書構成のサイトに到達しても問題ないはずです。

Canary スクリプト用のライブラリ関数

CloudWatch Synthetics には複数の組み込みクラスおよび関数が用意されており、これらを Node.js スクリプトの作成時に呼び出して Canary として使用できます。

UI Canary と API Canary の両方に適用されるものもあります。他の関数は UI Canary にのみ適用されます。UI Canary は `getPage()` 関数を使用する Canary であり、Puppeteer をウェブドライバーとして使用することで、ウェブページをナビゲートしたり操作したりします。

> **Note**
>
> Canary をアップグレードして新しいバージョンの Synthetics ランタイムを使用するたびに、Canary が使用するすべての Synthetics ライブラリ関数も Synthetics ランタイムがサポートするのと同じバージョンの NodeJS に自動的にアップグレードされます。

トピック

- Node.js Canary スクリプト用のライブラリ関数
- Selenium を使用する Python Canary スクリプトで利用可能なライブラリ関数

Node.js Canary スクリプト用のライブラリ関数

このセクションでは、Node.js Canary スクリプトで使用できるライブラリ関数の一覧を示しています。

トピック

- すべての Canary に適用される Node.js ライブラリクラスおよび関数
- UI Canary のみに適用される Node.js ライブラリクラスおよび関数

- API Canary のみに適用されるライブラリクラスおよび関数

すべての Canary に適用される Node.js ライブラリクラスおよび関数

以下の Node.js 用の CloudWatch Synthetics ライブラリ関数は、すべての Canary について有用です。

トピック

- Synthetics クラス
- SyntheticsConfiguration クラス
- Synthetics logger
- SyntheticsLogHelper クラス

Synthetics クラス

すべての Canary の次の関数は、Synthetics クラスにあります。

addExecutionError(errorMessage, ex);

`errorMessage` はエラーの詳細を示し、`ex` は発生した例外を示します

`addExecutionError` を使用すると、Canary の実行エラーを設定できます。これにより、スクリプトの実行を中断することなく Canary を失敗させることができます。また、`successPercent` メトリクスにも影響を与えません。

Canary スクリプトの成功または失敗を確認するために重要でない場合にのみ、エラーを実行エラーとして追跡するようにします。

`addExecutionError` の使用例を次に示します。ここでは、エンドポイントの可用性をモニターリングしながら、ページが読み込まれた後にスクリーンショットを取得しています。スクリーンショットの作成に失敗したとしても、エンドポイントの可用性が判断されるわけではないため、スクリーンショットの作成中に発生したすべてのエラーは、キャッチした上で実行エラーとして追加します。可用性に関するメトリックスには、依然としてエンドポイントが起動し実行中であることが示されていますが、Canary のステータスは失敗としてマークされています。次に示すコードブロックのサンプルでは、このようなエラーをキャッチし、実行エラーとして追加します。

```
try {
        await synthetics.takeScreenshot(stepName, "loaded");
    } catch(ex) {
```

```
        synthetics.addExecutionError('Unable to take screenshot ', ex);
    }
```

getCanaryName();

Canary の名前を返します。

getCanaryArn();

Canary の ARN を返します。

getCanaryUserAgentString();

canary のカスタムユーザーエージェントを返します。

getRuntimeVersion();

この関数は、ランタイムバージョン syn-nodejs-puppeteer-3.0 以降で使用できます。これは、Canary の Synthetics ランタイムバージョンを返します。例えば、戻り値は syn-nodejs-puppeteer-3.0 になる可能性があります。

getLogLevel ();

Synthetics ライブラリの現在のログレベルを取得します。取り得る値には以下のものがあります。

- 0 – デバッグ
- 1 – 情報
- 2 – 警告
- 3 – エラー

例:

```
let logLevel = synthetics.getLogLevel();
```

setLogLevel();

Synthetics ライブラリのログレベルを設定します。取り得る値には以下のものがあります。

- 0 – デバッグ
- 1 – 情報

- 2 – 警告
- 3 – エラー

例:

```
synthetics.setLogLevel(0);
```

SyntheticsConfiguration クラス

このクラスは、syn-nodejs-2.1 ランタイムバージョン以降でのみ使用できます。

SyntheticsConfiguration クラスを使用して、Synthetics ライブラリ関数の動作を設定できます。例えば、このクラスを使用して、スクリーンショットをキャプチャしないように executeStep() 関数を設定できます。

CloudWatch Synthetics の設定をグローバルレベルで設定できます。これは、Canary のすべてのステップに適用されます。設定キーと値のペアを渡して、これらの設定をステップレベルで上書きすることもできます。

ステップレベルでオプションを渡すことができます。例については、「async executeStep(stepName, functionToExecute, [stepConfig]);」および「executeHttpStep(stepName, requestOptions, [callback], [stepConfig])」を参照してください

関数の定義:

setConfig(options)

options はオブジェクトです。これは Canary の設定可能なオプションのセットです。以下のセクションでは、*options* で使用し得るフィールドについて説明します。

すべての Canary の setConfig(options)

syn-nodejs-puppeteer-3.2 以降の Canary では、setConfig の (オプション) に、次のパラメータを含めることができます。

- includeRequestHeaders (ブール値) – レポートにリクエストヘッダーを含めるかどうか。デフォルト: false。
- includeResponseHeaders (ブール値) – レポートにレスポンスヘッダーを含めるかどうか。デフォルト: false。

- restrictedHeaders(配列) – ヘッダーが含まれている場合に無視するヘッダー値のリスト。これは、リクエストヘッダーとレスポンスヘッダーの両方に適用されます。例えば、includeRequestHeaders を true として、restrictedHeaders を ['Authorization'] として渡すことで、認証情報を非表示にすることができます。
- includeRequestBody (ブール値) – レポートにリクエスト本文を含めるかどうか。デフォルト: false。
- includeResponseBody (ブール値) – レポートにレスポンス本文を含めるかどうか。デフォルト: false。

CloudWatch メトリクスに関する setConfig (オプション)

syn-nodejs-puppeteer-3.1 以降を使用する Canary の場合、setConfig の (options) には、Canary によって発行されるメトリクスを決定する次のブール値パラメータを含めることができます。これらの各オプションのデフォルトは true です。aggregated で始まるオプションは、CanaryName ディメンションなしでメトリクスを出力するかどうかを決定します。これらのメトリクスを使用して、すべての Canary の集計結果を確認できます。その他のオプションは、CanaryName ディメンションを含むメトリクスを出力するかどうかを決定します。これらのメトリクスを使用して、個々の Canary の結果を確認できます。

Canary から発行される CloudWatch メトリクスのリストについては、「Canary によって発行される CloudWatch メトリクス」を参照してください。

- failedCanaryMetric (ブール値) – この Canary の Failed メトリクス (CanaryName ディメンションを含む) を出力するかどうか。デフォルト: true。
- failedRequestsMetric (ブール値) – この Canary の Failed requests メトリクス (CanaryName ディメンションを含む) を出力するかどうか。デフォルト: true。
- _2xxMetric (ブール値) – この Canary の 2xx メトリクス (CanaryName ディメンションを含む) を出力するかどうか。デフォルト: true。
- _4xxMetric (ブール値) – この Canary の 4xx メトリクス (CanaryName ディメンションを含む) を出力するかどうか。デフォルト: true。
- _5xxMetric (ブール値) – この Canary の 5xx メトリクス (CanaryName ディメンションを含む) を出力するかどうか。デフォルト: true。
- stepDurationMetric (ブール値) – この Canary の Step duration メトリクス (CanaryName StepName ディメンションを含む) を出力するかどうか。デフォルト: true。
- stepSuccessMetric (ブール値) – この Canary の Step success メトリクス (CanaryName StepName ディメンションを含む) を出力するかどうか。デフォルト: true。

- aggregatedFailedCanaryMetric (ブール値) – この Canary の Failed メトリクス (CanaryName ディメンションを含まない) を出力するかどうか。デフォルト: true。
- aggregatedFailedRequestsMetric (ブール値) – この Canary の Failed Requests メトリクス (CanaryName ディメンションを含まない) を出力するかどうか。デフォルト: true。
- aggregated2xxMetric (ブール値) – この Canary の 2xx メトリクス (CanaryName ディメンションを含まない) を出力するかどうか。デフォルト: true。
- aggregated4xxMetric (ブール値) – この Canary の 4xx メトリクス (CanaryName ディメンションを含まない) を出力するかどうか。デフォルト: true。
- aggregated5xxMetric (ブール値) – この Canary の 5xx メトリクス (CanaryName ディメンションを含まない) を出力するかどうか。デフォルト: true。
- visualMonitoringSuccessPercentMetric (ブール値) – この Canary の visualMonitoringSuccessPercent メトリクスを出力するかどうか。デフォルト: true。
- visualMonitoringTotalComparisonsMetric (ブール値) – この Canary の visualMonitoringTotalComparisons メトリクスを出力するかどうか。デフォルト: false。
- stepsReport(ブール値) — ステップ実行サマリーをレポートするかどうか。デフォルト: true。
- includeUrlPassword(ブール値) — URL に表示されるパスワードを含めるかどうか。デフォルトでは、機密データの開示を防ぐために、URL に表示されるパスワードはログとレポートで墨消しされます。デフォルト: false。
- restrictedUrlParameters (配列)— 編集する URL パスまたはクエリパラメータのリスト。これは、ログ、レポート、エラーに表示される URL に適用されます。パラメータ名では大文字と小文字が区別されます。アスタリスク (*) を値として渡すと、すべての URL パスおよびクエリパラメータ値を墨消しできます。デフォルトは、空白の配列です。
- logRequest (ブール値) -すべてのリクエストを Canary ログに記録するかどうか。UI Canary の場合、ブラウザから送信された各リクエストがログに記録されます。デフォルト: true。
- logResponse (ブール値) -すべてのレスポンスを Canary ログに記録するかどうか。UI Canary の場合、ブラウザが受信したすべてのレスポンスをログに記録します。デフォルト: true。
- logRequestBody (ブール値) -リクエスト本文を Canary ログのリクエストとともに記録するかどうか。この設定は、logRequest が true である場合にのみ適用されます。デフォルト: false。
- logResponseBody (ブール値) -レスポンス本文を Canary ログのレスポンスとともに記録するかどうか。この設定は、logResponse が true である場合にのみ適用されます。デフォルト: false。

- `logRequestHeaders` (ブール値) -リクエストヘッダーを Canary ログのリクエストとともに記録するかどうか。この設定は、`logRequest` が `true` である場合にのみ適用されます。デフォルト: `false`。

  `includeRequestHeaders` は、アーティファクトのヘッダーを有効にすることに注意してください。
- `logResponseHeaders` (ブール値) -レスポンスヘッダーを Canary ログのレスポンスとともに記録するかどうか。この設定は、`logResponse` が `true` である場合にのみ適用されます。デフォルト: `false`。

  `includeResponseHeaders` は、アーティファクトのヘッダーを有効にすることに注意してください。

> **Note**
>
> `Duration` メトリクスと `SuccessPercent` メトリクスは、`CanaryName` メトリクスの有無にかかわらず、常に各 Canary について出力されます。

メトリクスを有効または無効にする方法

disableAggregatedRequestMetrics()

Canary が `CanaryName` ディメンションなしで出力されるすべてのリクエストメトリクスを出力するのを無効にします。

disableRequestMetrics()

Canary ごとのメトリクスとすべての Canary で集計されたメトリクスの両方を含む、すべてのリクエストメトリクスを無効にします。

disableStepMetrics()

ステップ成功メトリクスとステップ期間メトリクスの両方を含む、すべてのステップメトリクスを無効にします。

enableAggregatedRequestMetrics()

Canary が `CanaryName` ディメンションなしで出力されるすべてのリクエストメトリクスを送信できるようにします。

enableRequestMetrics()

Canary ごとのメトリクスとすべての Canary で集計されたメトリクスの両方を含む、すべてのリクエストメトリクスを有効にします。

enableStepMetrics()

ステップ成功メトリクスとステップ期間メトリクスの両方を含む、すべてのステップメトリクスを有効にします。

get2xxMetric()

Canary が `CanaryName` ディメンションを含む `2xx` メトリクスを出力するかどうかを返します。

get4xxMetric()

Canary が `CanaryName` ディメンションを含む `4xx` メトリクスを出力するかどうかを返します。

get5xxMetric()

Canary が `CanaryName` ディメンションを含む `5xx` メトリクスを出力するかどうかを返します。

getAggregated2xxMetric()

Canary がディメンションを含まない `2xx` メトリクスを出力するかどうかを返します。

getAggregated4xxMetric()

Canary がディメンションを含まない `4xx` メトリクスを出力するかどうかを返します。

getAggregatedFailedCanaryMetric()

Canary がディメンションを含まない `Failed` メトリクスを出力するかどうかを返します。

getAggregatedFailedRequestsMetric()

Canary がディメンションを含まない `Failed requests` メトリクスを出力するかどうかを返します。

getAggregated5xxMetric()

Canary がディメンションを含まない `5xx` メトリクスを出力するかどうかを返します。

getFailedCanaryMetric()

Canary が CanaryName ディメンションを含む Failed メトリクスを出力するかどうかを返します。

getFailedRequestsMetric()

Canary が CanaryName ディメンションを含む Failed requests メトリクスを出力するかどうかを返します。

getStepDurationMetric()

Canary がこの Canary に対して CanaryName ディメンションを含む Duration メトリクスを出力するかどうかを返します。

getStepSuccessMetric()

Canary がこの Canary に対して CanaryName ディメンションを含む StepSuccess メトリクスを出力するかどうかを返します。

with2xxMetric(_2xxMetric)

ブール引数を受け入れます。この引数は、この Canary に対して 2xx ディメンションを持つ CanaryName メトリクスを出力するかどうかを指定します。

with4xxMetric(_4xxMetric)

ブール引数を受け入れます。この引数は、この Canary に対して 4xx ディメンションを持つ CanaryName メトリクスを出力するかどうかを指定します。

with5xxMetric(_5xxMetric)

ブール引数を受け入れます。この引数は、この Canary に対して 5xx ディメンションを持つ CanaryName メトリクスを出力するかどうかを指定します。

withAggregated2xxMetric(aggregated2xxMetric)

ブール引数を受け入れます。この引数は、この Canary に対してディメンションを持たない 2xx メトリクスを出力するかどうかを指定します。

withAggregated4xxMetric(aggregated4xxMetric)

ブール引数を受け入れます。この引数は、この Canary に対してディメンションを持たない 4xx メトリクスを出力するかどうかを指定します。

withAggregated5xxMetric(aggregated5xxMetric)

ブール引数を受け入れます。この引数は、この Canary に対してディメンションを持たない 5xx メトリクスを出力するかどうかを指定します。

withAggregatedFailedCanaryMetric(aggregatedFailedCanaryMetric)

ブール引数を受け入れます。この引数は、この Canary に対してディメンションを持たない Failed メトリクスを出力するかどうかを指定します。

withAggregatedFailedRequestsMetric(aggregatedFailedRequestsMetric)

ブール引数を受け入れます。この引数は、この Canary に対してディメンションを持たない Failed requests メトリクスを出力するかどうかを指定します。

withFailedCanaryMetric(failedCanaryMetric)

ブール引数を受け入れます。この引数は、この Canary に対して Failed ディメンションを持つ CanaryName メトリクスを出力するかどうかを指定します。

withFailedRequestsMetric(failedRequestsMetric)

ブール引数を受け入れます。この引数は、この Canary に対して Failed requests ディメンションを持つ CanaryName メトリクスを出力するかどうかを指定します。

withStepDurationMetric(stepDurationMetric)

ブール引数を受け入れます。この引数は、この Canary に対して Duration ディメンションを持つ CanaryName メトリクスを出力するかどうかを指定します。

withStepSuccessMetric(stepSuccessMetric)

ブール引数を受け入れます。この引数は、この Canary に対して StepSuccess ディメンションを持つ CanaryName メトリクスを出力するかどうかを指定します。

他の機能を有効または無効にする方法

withHarFile()

この Canary の HAR ファイルを作成するかどうかを指定するブール値の引数を受け入れます。

withStepsReport()

この Canary のステップ実行サマリをレポートするかどうかを指定するブール値の引数を受け入れます。

withIncludeUrlPassword()

ログおよびレポートの URL に表示されるパスワードを含めるかどうかを指定するブール値の引数を受け入れます。

withRestrictedUrlParameters()

編集する URL パスまたはクエリパラメータの配列を受け入れます。これは、ログ、レポート、エラーに表示される URL に適用されます。アスタリスク (*) を値として渡すと、すべての URL パスおよびクエリパラメータ値を墨消しできます。

withLogRequest()

Canary のログにすべてのリクエストを記録するかどうかを指定するブール値の引数を受け入れます。

withLogResponse()

Canary のログにすべてのレスポンスを記録するかどうかを指定するブール値の引数を受け入れます。

withLogRequestBody()

Canary のログにすべてのリクエスト本文を記録するかどうかを指定するブール値の引数を受け入れます。

withLogResponseBody()

Canary のログにすべてのレスポンス本文を記録するかどうかを指定するブール値の引数を受け入れます。

withLogRequestHeaders()

Canary のログにすべてのリクエストヘッダーを記録するかどうかを指定するブール値の引数を受け入れます。

withLogResponseHeaders()

Canary のログにすべてのレスポンスヘッダーを記録するかどうかを指定するブール値の引数を受け入れます。

getHarFile()

Canary が HAR ファイルを作成するかどうかを返します。

getStepsReport()

Canary がステップ実行サマリをレポートするかどうかを返します。

getIncludeUrlPassword()

ログおよびレポートの URL に表示されるパスワードを、Canary に含めるかどうかを返します。

getRestrictedUrlParameters()

Canary が URL パスまたはクエリパラメータを編集するかどうかを返します。

getLogRequest()

Canary が Canary ログ内のすべてのリクエストを記録するかどうかを返します。

getLogResponse()

Canary が Canary ログ内のすべてのレスポンスを記録するかどうかを返します。

getLogRequestBody()

Canary が Canary ログ内のすべてのリクエスト本文を記録するかどうかを返します。

getLogResponseBody()

Canary が Canary ログ内のすべてのレスポンス本文を記録するかどうかを返します。

getLogRequestHeaders()

Canary が Canary ログ内のすべてのリクエストヘッダーを記録するかどうかを返します。

getLogResponseHeaders()

Canary が Canary ログ内のすべてのレスポンスヘッダーを記録するかどうかを返します。

すべての Canary の関数

- `withIncludeRequestHeaders`(includeRequestHeaders)
- `withIncludeResponseHeaders`(includeResponseHeaders)
- `withRestrictedHeaders`(restrictedHeaders)
- `withIncludeRequestBody`(includeRequestBody)
- `withIncludeResponseBody`(includeResponseBody)
- `enableReportingOptions`() – すべてのレポートオプションを有効にします-- includeRequestHeaders、includeResponseHeaders、includeRequestBody、および includeResponseBody。
- `disableReportingOptions`() – すべてのレポートオプションを無効にします-- includeRequestHeaders、includeResponseHeaders、includeRequestBody、および includeResponseBody。

UI Canary の setConfig(options)

UI Canary の場合、setConfig には次のブール値のパラメータを含めることができます。

- `continueOnStepFailure` (ブール値) – ステップが失敗した後も Canary スクリプトの実行を続行するかどうか (これは executeStep 関数を参照します)。いずれかのステップが失敗しても、Canary 実行は失敗としてマークされます。デフォルト: `false`。
- `harFile`(ブール値) — HAR ファイルを作成するかどうか。デフォルト: `True`。
- `screenshotOnStepStart` (ブール値) – ステップを開始する前にスクリーンショットを作成するかどうか。
- `screenshotOnStepSuccess` (ブール値) – ステップが正常に完了した後にスクリーンショットを作成するかどうか。
- `screenshotOnStepFailure` (ブール値) – ステップが失敗した後にスクリーンショットを作成するかどうか。

スクリーンショットを有効または無効にする方法

disableStepScreenshots()

すべてのスクリーンショットオプション (screenshotOnStepStart、screenshotOnStepSuccess、screenshotOnStepFailure) を無効にします。

enableStepScreenshots()

すべてのスクリーンショットオプション (screenshotOnStepStart、screenshotOnStepSuccess、screenshotOnStepFailure) を有効にします。デフォルトでは、これらすべてのメソッドが有効になります。

getScreenshotOnStepFailure()

ステップが失敗した後、Canary がスクリーンショットを作成するかどうかを返します。

getScreenshotOnStepStart()

Canary がステップを開始する前にスクリーンショットを作成するかどうかを返します。

getScreenshotOnStepSuccess()

Canary がステップを正常に完了した後にスクリーンショットを作成するかどうかを返します。

withScreenshotOnStepStart(screenshotOnStepStart)

ステップを開始する前にスクリーンショットを作成するかどうかを示すブール値の引数を受け入れます。

withScreenshotOnStepSuccess(screenshotOnStepSuccess)

ステップを正常に完了した後にスクリーンショットを作成するかどうかを示すブール値の引数を受け入れます。

withScreenshotOnStepFailure(screenshotOnStepFailure)

ステップが失敗した後にスクリーンショットを作成するかどうかを示すブール値の引数を受け入れます。

UI Canary での使用

まず、Synthetics 依存関係をインポートし、設定を取得します。

```
// Import Synthetics dependency
const synthetics = require('Synthetics');

// Get Synthetics configuration
const synConfig = synthetics.getConfiguration();
```

次に、以下のいずれかのオプションを使用して setConfig メソッドを呼び出すことで、各オプションの設定を行います。

```
// Set configuration values
    synConfig.setConfig({
        screenshotOnStepStart: true,
        screenshotOnStepSuccess: false,
        screenshotOnStepFailure: false
    });
```

または

```
synConfig.withScreenshotOnStepStart(false).withScreenshotOnStepSuccess(true).withScreenshotOnSt
```

すべてのスクリーンショットを無効にするには、次の例のように disableStepScreenshots() 関数を使用します。

```
synConfig.disableStepScreenshots();
```

コード内の任意の個所でスクリーンショットを有効または無効にすることができます。例えば、1 つのステップでのみスクリーンショットを無効にするには、そのステップを実行する前にスクリーンショットを無効にしてステップの実行後に有効にします。

API Canary の setConfig(options)

API Canary の場合、setConfig には次のブール値のパラメータを含めることができます。

- `continueOnHttpStepFailure` (ブール値) – HTTP ステップが失敗した後も Canary スクリプトの実行を続行するかどうか (これは executeHttpStep 関数を参照します)。いずれかのステップが失敗しても、Canary 実行は失敗としてマークされます。デフォルト: `true`。

ビジュアルモニターリング

ビジュアルモニターリングは、Canary 実行中に撮影されたスクリーンショットと、ベースライン Canary 実行中に撮影されたスクリーンショットを比較します。2 つのスクリーンショットの不一致がしきい値のパーセンテージを超えると、Canary は失敗し、Canary 実行レポートで色の違いが強調表示されている領域を確認できます。ビジュアルモニターリングは、syn-puppeteer-node-3.2 以降で実行中の Canary でサポートされています。Python と Selenium を実行している Canary では現在サポートされていません。

ビジュアルモニターリングを有効にするには、Canary スクリプトに次のコード行を追加します。詳細については、「SyntheticsConfiguration クラス」を参照してください。

```
syntheticsConfiguration.withVisualCompareWithBaseRun(true);
```

この行がスクリプトに追加された後に初めて Canary が正常に実行されると、その実行中に撮影されたスクリーンショットが比較のベースラインとして使用されます。最初に Canary を実行した後、CloudWatch コンソールを使用して Canary を編集して、次のいずれかの操作を行うことができます。

- Canary の次の実行を新しいベースラインとして設定する。
- 現在のベースラインスクリーンショットに境界を描画し、ビジュアル比較時に無視するスクリーンショットの領域を指定する。
- スクリーンショットをビジュアルモニターリングに使用しないようにする。

CloudWatch コンソールを使用して Canary を編集する方法の詳細については、「Canary を編集または削除する」を参照してください。

ビジュアルモニターリングのその他のオプション

syntheticsConfiguration.withVisualVarianceThresholdPercentage(desiredPercentage)

ビジュアル比較におけるスクリーンショットの差異の許容パーセンテージを設定します。

syntheticsConfiguration.withVisualVarianceHighlightHexColor("#fafa00")

ビジュアルモニターリングを使用する Canary 実行レポートを表示するときに、分散領域を指定するハイライト色を設定します。

syntheticsConfiguration.withFailCanaryRunOnVisualVariance(failCanary)

しきい値を超える視覚的な違いがある場合に、Canary が失敗するかどうかを設定します。デフォルトでは、Canary は失敗します。

Synthetics logger

SyntheticsLogger は、コンソールと、同じログレベルのローカルログファイルの両方にログを書き込みます。このログファイルは、ログレベルが、呼び出されたログ関数の該当するログ記録レベル以下である場合にのみ、両方の場所に書き込まれます。

ローカルログファイルのログ記録ステートメントには、呼び出された関数のログレベルに合わせて「DEBUG:」や「INFO:」などが先頭に追加されます。

SyntheticsLogger は、Synthetics Canary ログ記録と同じログレベルで Synethetics ライブラリを実行する場合に使用できます。

S3 の結果の場所にアップロードされるログファイルを作成する場合、SyntheticsLogger を使用する必要はありません。代わりに、別のログファイルを /tmp フォルダ内に作成できます。/tmp フォルダ内に作成したファイルは、アーティファクトとして S3 の結果の場所にアップロードされます。

Synthetics Library logger を使用するには:

```
const log = require('SyntheticsLogger');
```

有用な関数定義:

log.debug(***message***, ***ex***);

パラメータ: *message* はログに記録するメッセージです。*ex* はログに記録する例外 (ある場合) です。

例:

```
log.debug("Starting step - login.");
```

log.error(***message***, ***ex***);

パラメータ: *message* はログに記録するメッセージです。*ex* はログに記録する例外 (ある場合) です。

例:

```
try {
  await login();
catch (ex) {
  log.error("Error encountered in step - login.", ex);
}
```

log.info(***message***, ***ex***);

パラメータ: *message* はログに記録するメッセージです。*ex* はログに記録する例外 (ある場合) です。

例:

```
log.info("Successfully completed step - login.");
```

log.log(***message***, ***ex***);

これは log.info のエイリアスです。

パラメータ: *message* はログに記録するメッセージです。*ex* はログに記録する例外 (ある場合) です。

例:

```
 log.log("Successfully completed step - login.");
```

log.warn(***message***, ***ex***);

パラメータ: *message* はログに記録するメッセージです。*ex* はログに記録する例外 (ある場合) です。

例:

```
log.warn("Exception encountered trying to publish CloudWatch Metric.", ex);
```

SyntheticsLogHelper クラス

SyntheticsLogHelper クラスは、ランタイム syn-nodejs-puppeteer-3.2 以降で利用可能です。これは CloudWatch Synthetics ライブラリですでに初期化されており、Synthetics 構成で設定されています。スクリプトに依存関係としてこれを追加できます。このクラスを使用すると、URL、ヘッダー、およびエラーメッセージをサニタイズして、機密情報を編集できます。

> **Note**
>
> Synthetics は、Synthetics の構成設定 restrictedUrlParameters に基づいて、ログ、レポート、HAR ファイル、および Canary 実行エラーに含める前に、ログに記録されるすべての URL とエラーメッセージをサニタイズします。getSanitizedUrl または

getSanitizedErrorMessage は、スクリプトに URL またはエラーを記録している場合にのみ使用できます。Synthetics は、スクリプトによってスローされた Canary エラーを除いて、Canary アーティファクトを保存しません。Canary 実行アーティファクトは、顧客アカウントに保存されます。詳細については、「Synthetics Canary のセキュリティ上の考慮事項」を参照してください。

getSanitizedUrl(url, stepConfig = null)

この関数は、syn-nodejs-puppeteer-3.2 以降で使用できます。これは、設定に基づいてサニタイズされた URL 文字列を返します。プロパティ restrictedUrlParameters1を設定することで、パスワードや access_token などの機密性の高い URL を編集するように選択できます。デフォルトでは、URL 内のパスワードは編集できます。URL パスワードを有効にするには、必要に応じて includeUrlPassword を true に設定します。

渡された URL が有効な URL でない場合、この関数は、エラーをスローします。

パラメータ

- *url* は文字列で、サニタイズする URL です。
- *stepConfig* (オプション) この関数のグローバル Synthetics 設定を上書きします。もし stepConfig が渡されない場合、グローバル設定を使用して URL をサニタイズします。

例

この例では、次のサンプル URL を使用しています: https://example.com/learn/home?access_token=12345&token_type=Bearer&expires_in=1200。この例では、access_token には、記録すべきではない機密情報が含まれています。Synthetics サービスは、Canary 実行 Artifact を保存しないことに注意してください。ログ、スクリーンショット、レポートなどの Artifact は、すべてカスタマーアカウントの Amazon S3 バケットに保存されます。

最初のステップでは、Synthetics の構成を設定します。

```
// Import Synthetics dependency
const synthetics = require('Synthetics');

// Import Synthetics logger for logging url
const log = require('SyntheticsLogger');
```

```
// Get Synthetics configuration
const synConfig = synthetics.getConfiguration();

// Set restricted parameters
synConfig.setConfig({
   restrictedUrlParameters: ['access_token'];
});
```

次に、URL をサニタイズしてログに記録します

```
// Import SyntheticsLogHelper dependency
const syntheticsLogHelper = require('SyntheticsLogHelper');

const sanitizedUrl = synthetics.getSanitizedUrl('https://example.com/learn/home?
access_token=12345&token_type=Bearer&expires_in=1200');
```

これは、Canary ログに次のように記録されます。

```
My example url is: https://example.com/learn/home?
access_token=REDACTED&token_type=Bearer&expires_in=1200
```

次の例のように、Synthetics 設定オプションを含むオプションのパラメータを渡すことによって、URL の Synthetics 設定を上書きできます。

```
const urlConfig = {
   restrictedUrlParameters = ['*']
};
const sanitizedUrl = synthetics.getSanitizedUrl('https://example.com/learn/home?
access_token=12345&token_type=Bearer&expires_in=1200', urlConfig);
logger.info('My example url is: ' + sanitizedUrl);
```

上記の例では、すべてのクエリパラメータを編集し、次のように記録されています。

```
My example url is: https://example.com/learn/home?
access_token=REDACTED&token_type=REDACTED&expires_in=REDACTED
```

getSanitizedErrorMessage

この関数は、syn-nodejs-puppeteer-3.2 以降で使用できます。これは、Synthetics の設定に基づいて存在するすべての URL をサニタイズすることによって、サニタイズされたエラー文字列を返

します。この関数を呼び出すときに、オプションのstepConfig パラメータを渡すことで、グローバル Synthetics 設定をオーバーライドするように選択できます。

パラメータ

- *error* はサニタイズするエラーです。Error オブジェクトまたは文字列にすることができます。
- *stepConfig* (オプション) この関数のグローバル Synthetics 設定を上書きします。もし stepConfig が渡されない場合、グローバル設定を使用して URL をサニタイズします。

例

この例では、次のエラーを使用しています: `Failed to load url: https://example.com/learn/home?access_token=12345&token_type=Bearer&expires_in=1200`

最初のステップでは、Synthetics の構成を設定します。

```
// Import Synthetics dependency
const synthetics = require('Synthetics');

// Import Synthetics logger for logging url
const log = require('SyntheticsLogger');

// Get Synthetics configuration
const synConfig = synthetics.getConfiguration();

// Set restricted parameters
synConfig.setConfig({
   restrictedUrlParameters: ['access_token'];
});
```

次に、サニタイズしてエラーメッセージを口グに記録します

```
// Import SyntheticsLogHelper dependency
const syntheticsLogHelper = require('SyntheticsLogHelper');

try {
   // Your code which can throw an error containing url which your script logs
} catch (error) {
    const sanitizedErrorMessage = synthetics.getSanitizedErrorMessage(errorMessage);
    logger.info(sanitizedErrorMessage);
}
```

これは、Canary ログに次のように記録されます。

```
Failed to load url: https://example.com/learn/home?
access_token=REDACTED&token_type=Bearer&expires_in=1200
```

getSanitizedHeaders(headers, stepConfig=null)

この関数は、syn-nodejs-puppeteer-3.2 以降で使用できます。これは、syntheticsConfiguration の restrictedHeaders プロパティに基づいてサニタイズされたヘッダーを返します。restrictedHeaders プロパティで指定されたヘッダーは、ログ、HAR ファイル、およびレポートから編集されます。

パラメータ

- *headers* は、サニタイズするヘッダーを含むオブジェクトです。
- *stepConfig* (オプション) この関数のグローバル Synthetics 設定を上書きします。stepConfig が渡されない場合、グローバル設定を使用してヘッダーをサニタイズします。

UI Canary のみに適用される Node.js ライブラリクラスおよび関数

Node.js 用の次の CloudWatch Synthetics ライブラリ関数は、UI の Canary にのみ適用されます。

トピック

- Synthetics クラス
- BrokenLinkCheckerReport クラス
- SyntheticsLink クラス

Synthetics クラス

Synthetics クラスには、次の関数が含まれます。

async addUserAgent(page, userAgentString);

この関数は、指定されたページの user-agent ヘッダーに *userAgentString* を追加します。

例:

```
await synthetics.addUserAgent(page, "MyApp-1.0");
```

ページの user-agent ヘッダーが *browsers-user-agent-header-value*MyApp-1.0 に設定されます。

async executeStep(stepName, functionToExecute, [stepConfig]);

指定されたステップを実行します。ステップは、開始/合格/失敗の口グ記録、開始/合格/失敗のスクリーンショット、合格/失敗と所要時間のメトリクスでラップされます。

> **Note**
>
> syn-nodejs-2.1 以降のランタイムを使用している場合は、スクリーンショットを作成するかどうかと、作成するタイミングを設定できます。詳細については、「SyntheticsConfiguration クラス」を参照してください。

executeStep 関数は、次のことも行います。

- ステップが開始されたことを口グに記録します。
- <stepName>-starting という名前でスクリーンショットを作成します。
- タイマーを開始します。
- 指定された関数を実行します。
- 関数が正常に戻ると、合格としてカウントします。関数がスローされると、失敗としてカウントします。
- タイマーを終了します。
- ステップが合格したか失敗したかを口グに記録します。
- <stepName>-succeeded や <stepName>-failed という名前でスクリーンショットを作成します。
- stepName SuccessPercent メトリクスを出力します。合格の場合は 100、不合格の場合は 0 です。
- stepName Duration メトリクスを出力します。値は、ステップの開始時刻と終了時刻に基づきます。
- 最後に、functionToExecute から返された内容を返し、functionToExecute からスローされた内容を再スローします。

Canary が syn-nodejs-2.0 ランタイム以降を使用する場合、この関数はステップ実行の要約を Canary のレポートに追加します。要約には、開始時刻、終了時刻、ステータス (PASSED/

FAILED)、エラーの理由 (エラーの場合) 、各ステップの実行中にキャプチャされたスクリーンショットなど、各ステップの詳細が含まれます。

例:

```
await synthetics.executeStep('navigateToUrl', async function (timeoutInMillis = 30000)
 {
           await page.goto(url, {waitUntil: ['load', 'networkidle0'], timeout:
 timeoutInMillis});});
```

レスポンス:

`functionToExecute` から返された内容を返します。

syn-nodejs-2.2 を使用した更新

`syn-nodejs-2.2` を使用して開始し、オプションでステップ設定を渡して、ステップレベルで CloudWatch Synthetics 設定を上書きできます。`executeStep` に渡すことができるオプションのリストについては、「SyntheticsConfiguration クラス」を参照してください。

次の例では、`continueOnStepFailure` のデフォルトの `false` 設定を `true` に上書きし、スクリーンショットをいつ取得するかを指定します。

```
var stepConfig = {
    'continueOnStepFailure': true,
    'screenshotOnStepStart': false,
    'screenshotOnStepSuccess': true,
    'screenshotOnStepFailure': false
}

await executeStep('Navigate to amazon', async function (timeoutInMillis = 30000) {
      await page.goto(url, {waitUntil: ['load', 'networkidle0'], timeout:
 timeoutInMillis});
 }, stepConfig);
```

getDefaultLaunchOptions();

`getDefaultLaunchOptions()` 関数は、CloudWatch Synthetics で使用するブラウザ起動オプションを返します。詳細については、「起動オプションの種類」を参照してください

```
// This function returns default launch options used by Synthetics.
```

```
const defaultOptions = await synthetics.getDefaultLaunchOptions();
```

getPage ();

現在開いているページを Puppeteer オブジェクトとして返します。詳細については、Puppeteer API v1.14.0 を参照してください。

例:

```
let page = synthetics.getPage();
```

レスポンス:

現在のブラウザセッションで現在開いているページ (Puppeteer オブジェクト)。

getRequestResponseLogHelper();

> **Important**
>
> syn-nodejs-puppeteer-3.2 以降のランタイムを使用する Canary で、この関数は RequestResponseLogHelper クラスと共に廃止予定です。この関数を使用すると、Canary ログに警告が表示されます。この関数は、将来のランタイムバージョンで削除されます。この関数を使用している場合は、代わりに RequestResponseLogHelper クラス を使用してください。

この関数は、リクエストとレスポンスのログ記録フラグを調整するためのビルダーパターンとして使用します。

例:

```
synthetics.setRequestResponseLogHelper(getRequestResponseLogHelper().withLogRequestHeaders(fals
```

レスポンス:

```
{RequestResponseLogHelper}
```

launch(options)

この関数のオプションは、syn-nodejs-2.1 ランタイムバージョン以降でのみ使用できます。

この関数は UI Canary でのみ使用します。これは、既存のブラウザを閉じ、新しいブラウザを起動します。

> **Note**
>
> CloudWatch Synthetics は、スクリプトの実行を開始する前に、必ずブラウザを起動します。カスタムオプションを使用して新しいブラウザを起動する場合を除いては、launch() を呼び出す必要はありません。

(options) は、ブラウザで設定する構成可能なオプションのセットです。詳細については、「起動オプションの種類」を参照してください。

この関数をオプションなしで呼び出すと、Synthetics はデフォルトの引数である `executablePath` と `defaultViewport` を使用してブラウザを起動します。CloudWatch Synthetics のデフォルトのビューポートは 1920 x 1080 です。

ブラウザを起動するときに、CloudWatch Synthetics で使用されている起動パラメータを上書きし、追加のパラメータを渡すことができます。例えば、次のコードスニペットでは、デフォルトの引数とデフォルトの実行可能パスを使用してブラウザを起動しますが、ビューポートは 800 x 600 になります。

```
await synthetics.launch({
        defaultViewport: {
            "deviceScaleFactor": 1,
            "width": 800,
            "height": 600
    }});
```

次のサンプルコードは、CloudWatch Synthetics の起動パラメータに新しい `ignoreHTTPSErrors` パラメータを追加します。

```
await synthetics.launch({
        ignoreHTTPSErrors: true
 });
```

ウェブセキュリティを無効にするには、CloudWatch Synthetics の起動パラメータの引数に `--disable-web-security` フラグを追加します。

```
// This function adds the --disable-web-security flag to the launch parameters
```

```
const defaultOptions = await synthetics.getDefaultLaunchOptions();
const launchArgs = [...defaultOptions.args, '--disable-web-security'];
await synthetics.launch({
     args: launchArgs
  });
```

RequestResponseLogHelper クラス

> **Important**
>
> syn-nodejs-puppeteer-3.2 以降のランタイムを使用する Canary で、このクラスは廃止予定です。このクラスを使用すると、Canary ログに警告が表示されます。この関数は、将来のランタイムバージョンで削除されます。この関数を使用している場合は、代わりに RequestResponseLogHelper クラス を使用してください。

リクエストおよびレスポンスペイロードの文字列表現の詳細な設定および作成を処理します。

```
class RequestResponseLogHelper {

    constructor () {
        this.request = {url: true, resourceType: false, method: false, headers: false,
 postData: false};
        this.response = {status: true, statusText: true, url: true, remoteAddress:
 false, headers: false};
    }

    withLogRequestUrl(logRequestUrl);

    withLogRequestResourceType(logRequestResourceType);

    withLogRequestMethod(logRequestMethod);

    withLogRequestHeaders(logRequestHeaders);

    withLogRequestPostData(logRequestPostData);


    withLogResponseStatus(logResponseStatus);

    withLogResponseStatusText(logResponseStatusText);
```

```
    withLogResponseUrl(logResponseUrl);

    withLogResponseRemoteAddress(logResponseRemoteAddress);

    withLogResponseHeaders(logResponseHeaders);
```

例:

```
synthetics.setRequestResponseLogHelper(getRequestResponseLogHelper()
.withLogRequestPostData(true)
.withLogRequestHeaders(true)
.withLogResponseHeaders(true));
```

レスポンス:

```
{RequestResponseLogHelper}
```

setRequestResponseLogHelper();

> **Important**
>
> `syn-nodejs-puppeteer-3.2` 以降のランタイムを使用する Canary で、この関数は `RequestResponseLogHelper` クラスと共に廃止予定です。この関数を使用すると、Canary ログに警告が表示されます。この関数は、将来のランタイムバージョンで削除されます。この関数を使用している場合は、代わりに RequestResponseLogHelper クラス を使用してください。

この関数は、リクエストとレスポンスのログ記録フラグを設定するためのビルダーパターンとして使用します。

例:

```
synthetics.setRequestResponseLogHelper().withLogRequestHeaders(true).withLogResponseHeaders(tru
```

レスポンス:

```
{RequestResponseLogHelper}
```

async takeScreenshot(name, suffix);

名前とサフィックス (オプション) を使用して現在のページのスクリーンショット (.PNG) を作成します。

例:

```
await synthetics.takeScreenshot("navigateToUrl", "loaded")
```

この例では、`01-navigateToUrl-loaded.png` という名前のスクリーンショットをキャプチャし、Canary の S3 バケットにアップロードします。

最初のパラメータとして `stepName` を渡すことにより、特定の Canary ステップのスクリーンショットを作成できます。スクリーンショットはレポートの Canary ステップにリンクされ、デバッグ中に各ステップを追跡するのに役立ちます。

CloudWatch Synthetics Canary は、ステップ (`executeStep` 関数) の開始前とステップ完了後に自動的にスクリーンショットを作成します (スクリーンショットを無効にするように Canary を設定している場合を除きます)。`takeScreenshot` 関数でステップ名を渡すことで、さらに多くのスクリーンショットを作成できます。

次の例では、`stepName` の値として `signupForm` を使用してスクリーンショットを作成します。スクリーンショットには `02-signupForm-address` という名前が付けられ、Canary レポートで指定された `signupForm` という名前のステップにリンクされます。

```
await synthetics.takeScreenshot('signupForm', 'address')
```

BrokenLinkCheckerReport クラス

このクラスは、Synthetics リンクを追加するメソッドを提供します。これは、`syn-nodejs-2.0-beta` 以降のバージョンのランタイムを使用する Canary でのみサポートされています。

`BrokenLinkCheckerReport` を使用するには、スクリプトに次の行を含めます。

```
const BrokenLinkCheckerReport = require('BrokenLinkCheckerReport');

const brokenLinkCheckerReport = new BrokenLinkCheckerReport();
```

有用な関数定義:

addLink(***syntheticsLink***, isBroken)

syntheticsLink は、リンクを表す SyntheticsLink オブジェクトです。この関数は、ステータスコードに従ってリンクを追加します。デフォルトでは、ステータスコードが利用できない場合、またはステータスコードが 400 以上の場合、リンク切れとみなされます。このデフォルトの動作は、値が true または false のオプションのパラメータ isBrokenLink を渡すことによって上書きできます。

この関数には戻り値がありません。

getLinks()

この関数は、リンク切れチェッカーレポートに含まれる SyntheticsLink オブジェクトの配列を返します。

getTotalBrokenLinks()

この関数は、リンク切れの総数を表す数値を返します。

getTotalLinksChecked()

この関数は、レポートに含まれるリンクの総数を表す数値を返します。

BrokenLinkCheckerReport の使用方法

次の Canary スクリプトコードスニペットは、リンクに移動して、リンク切れチェッカーレポートに追加する例を示しています。

1. SyntheticsLink、BrokenLinkCheckerReport、および Synthetics をインポートします。

```
const BrokenLinkCheckerReport = require('BrokenLinkCheckerReport');
const SyntheticsLink = require('SyntheticsLink');

// Synthetics dependency
const synthetics = require('Synthetics');
```

2. レポートにリンクを追加するには、BrokenLinkCheckerReport のインスタンスを作成します。

```
let brokenLinkCheckerReport = new BrokenLinkCheckerReport();
```

3. URL に移動し、リンク切れチェッカーレポートに追加します。

```
let url = "https://amazon.com";
```

```
let syntheticsLink = new SyntheticsLink(url);

// Navigate to the url.
let page = await synthetics.getPage();

// Create a new instance of Synthetics Link
let link = new SyntheticsLink(url)

try {
    const response = await page.goto(url, {waitUntil: 'domcontentloaded', timeout:
 30000});
} catch (ex) {
    // Add failure reason if navigation fails.
    link.withFailureReason(ex);
}

if (response) {
    // Capture screenshot of destination page
    let screenshotResult = await synthetics.takeScreenshot('amazon-home', 'loaded');

    // Add screenshot result to synthetics link
    link.addScreenshotResult(screenshotResult);

    // Add status code and status description to the link
    link.withStatusCode(response.status()).withStatusText(response.statusText())
}

// Add link to broken link checker report.
brokenLinkCheckerReport.addLink(link);
```

4. レポートを Synthetics に追加します。これにより、Canary 実行ごとに S3 バケットに `BrokenLinkCheckerReport.json` という名前の JSON ファイルが作成されます。コンソールには、各 Canary 実行のリンクレポートと、スクリーンショット、ログ、HAR ファイルを表示できます。

```
await synthetics.addReport(brokenLinkCheckerReport);
```

SyntheticsLink クラス

このクラスは、情報をラップするメソッドを提供します。これは、`syn-nodejs-2.0-beta` 以降のバージョンのランタイムを使用する Canary でのみサポートされています。

SyntheticsLink を使用するには、スクリプトに次の行を含めます。

```
const SyntheticsLink = require('SyntheticsLink');

const syntheticsLink = new SyntheticsLink("https://www.amazon.com");
```

この関数は、syntheticsLink*Object* を返します

有用な関数定義:

withUrl(***url***)

url は URL 文字列です。この関数は、syntheticsLink*Object* を返します

withText(***text***)

text はアンカーテキストを表す文字列です。この関数は syntheticsLink*Object* を返します。リンクに対応するアンカーテキストを追加します。

withParentUrl(***parentUrl***)

parentUrl は、親 (ソースページ) URL を表す文字列です。この関数は、syntheticsLink*Object* を返します

withStatusCode(***statusCode***)

statusCode は、ステータスコードを表す文字列です。この関数は、syntheticsLink*Object* を返します

withFailureReason(***failureReason***)

failureReason は、エラーの理由を表す文字列です。この関数は、syntheticsLink*Object* を返します

addScreenshotResult(***screenshotResult***)

screenshotResult はオブジェクトです。これは、Synthetics 関数 ScreenshotResult によって返された takeScreenshot のインスタンスです。このオブジェクトには以下が含まれています。

- fileName – screenshotFileName を表す文字列
- pageUrl (オプション)
- error (オプション)

API Canary のみに適用されるライブラリクラスおよび関数

以下の CloudWatch Synthetics ライブラリ関数は、API Canary についてのみ有用です。

トピック

- executeHttpStep(stepName, requestOptions, [callback], [stepConfig])

executeHttpStep(stepName, requestOptions, [callback], [stepConfig])

提供された HTTP リクエストをステップとして実行し、`SuccessPercent` (pass/fail) および `Duration` メトリクスを発行します。

executeHttpStep は、リクエストで指定されたプロトコルに応じて、HTTP または HTTPS ネイティブ関数のいずれかを内部で使用します。

この関数は、Canary のレポートにステップ実行の概要も追加します。概要には、次のような各 HTTP リクエストの詳細が含まれます。

- 開始時間
- 終了時間
- ステータス (PASSED/FAILED)
- 失敗の理由 (失敗した場合)
- リクエスト/レスポンスヘッダー、本文、ステータスコード、ステータスメッセージ、パフォーマンスタイミングなどの HTTP 呼び出しの詳細。

パラメータ

stepName(***###***)

ステップの名前を指定します。この名前は、このステップの CloudWatch メトリクスを発行する場合にも使用されます。

requestOptions(***############***)

このパラメータの値には、URL、URL 文字列、またはオブジェクトを指定できます。オブジェクトの場合、HTTP リクエストを行うために設定可能なオプションのセットである必要があります。Node.js ドキュメントの http.request(options[, callback]) のすべてのオプションをサポートしています。

これらの Node.js オプションに加えて、requestOptions では、追加のパラメータ body がサポートされます。body パラメータを使用して、データをリクエスト本文として渡すことができます。

callback(*#####*)

(オプション) これは HTTP レスポンスで呼び出されるユーザー関数です。レスポンスは、Class: http.IncomingMessage のタイプです。

stepConfig(*######*)

(オプション) このパラメータを使用して、このステップについて、異なる設定でグローバル Synthetics 設定を上書きします。

executeHttpStep の使用例

次の一連の例は、このオプションのさまざまな用途を説明するために、相互に関連しています。

この最初の例では、リクエストパラメータを設定します。requestOptions として URL を渡すことができます:

```
let requestOptions = 'https://www.amazon.com';
```

または、一連のオプションを渡すことができます:

```
let requestOptions = {
        'hostname': 'myproductsEndpoint.com',
        'method': 'GET',
        'path': '/test/product/validProductName',
        'port': 443,
        'protocol': 'https:'
    };
```

次の例では、レスポンスを受け入れるコールバック関数を作成します。デフォルトでは、コールバックを指定しない場合、CloudWatch Synthetics はステータスが 200 から 299 の範囲であることを検証します。

```
// Handle validation for positive scenario
    const callback = async function(res) {
        return new Promise((resolve, reject) => {
            if (res.statusCode < 200 || res.statusCode > 299) {
                throw res.statusCode + ' ' + res.statusMessage;
            }
```

```
            let responseBody = '';
            res.on('data', (d) => {
                responseBody += d;
            });

            res.on('end', () => {
                // Add validation on 'responseBody' here if required. For ex, your
 status code is 200 but data might be empty
                resolve();
            });
        });
    };
```

次の例では、グローバル CloudWatch Synthetics の設定を上書きするこのステップの設定を作成します。この例のステップ設定では、レポートでリクエストヘッダー、レスポンスヘッダー、リクエスト本文 (投稿データ)、およびレスポンス本文を許可し、「X-Amz-Security-Token」および「Authorization」ヘッダーの値を制限します。デフォルトでは、セキュリティ上の理由から、これらの値はレポートに含まれません。それらを含めるように選択した場合、データは S3 バケットにのみ保存されます。

```
// By default headers, post data, and response body are not included in the report for
 security reasons.
// Change the configuration at global level or add as step configuration for individual
 steps
let stepConfig = {
    includeRequestHeaders: true,
    includeResponseHeaders: true,
    restrictedHeaders: ['X-Amz-Security-Token', 'Authorization'], // Restricted header
 values do not appear in report generated.
    includeRequestBody: true,
    includeResponseBody: true
};
```

この最後の例では、リクエストを executeHttpStep に渡し、ステップに名前を付けます。

```
await synthetics.executeHttpStep('Verify GET products API', requestOptions, callback,
 stepConfig);
```

この一連の例では、CloudWatch Synthetics はレポートの各ステップの詳細を追加し、stepName を使用して各ステップのメトリクスを生成します。

`Verify GET products API` ステップの `successPercent` および `duration` メトリクスが表示されます。API 呼び出しステップのメトリクスをモニターリングすることで、API のパフォーマンスをモニターリングできます。

これらの関数を使用する完全なスクリプトのサンプルについては、「マルチステップ API Canary」を参照してください。

Selenium を使用する Python Canary スクリプトで利用可能なライブラリ関数

このセクションでは、Python Canary スクリプトで使用できる Selenium ライブラリ関数の一覧を示しています。

トピック

- すべての Canary に適用される Python および Selenium ライブラリクラスおよび関数
- UI Canary のみに適用される Python および Selenium ライブラリクラスおよび関数

すべての Canary に適用される Python および Selenium ライブラリクラスおよび関数

以下の Python 用の CloudWatch Synthetics Selenium ライブラリ関数は、すべての Canary について有用です。

トピック

- SyntheticsConfiguration クラス
- SyntheticsLogger クラス

SyntheticsConfiguration クラス

SyntheticsConfiguration クラスを使用して、Synthetics ライブラリ関数の動作を設定できます。例えば、このクラスを使用して、スクリーンショットをキャプチャしないように `executeStep()` 関数を設定できます。

CloudWatch Synthetics の設定は、グローバルレベルで設定できます。

関数の定義:

set_config(options)

```
from aws_synthetics.common import synthetics_configuration
```

options はオブジェクトです。これは Canary の設定可能なオプションのセットです。以下のセクションでは、*options* で使用し得るフィールドについて説明します。

- screenshot_on_step_start (ブール値) – ステップを開始する前にスクリーンショットを作成するかどうか。
- screenshot_on_step_success (ブール値) – ステップが正常に完了した後にスクリーンショットを作成するかどうか。
- screenshot_on_step_failure (ブール値) – ステップが失敗した後にスクリーンショットを作成するかどうか。

with_screenshot_on_step_start(screenshot_on_step_start)

ステップを開始する前にスクリーンショットを作成するかどうかを示すブール値の引数を受け入れます。

with_screenshot_on_step_success(screenshot_on_step_success)

ステップを正常に完了した後にスクリーンショットを作成するかどうかを示すブール値の引数を受け入れます。

with_screenshot_on_step_failure(screenshot_on_step_failure)

ステップが失敗した後にスクリーンショットを作成するかどうかを示すブール値の引数を受け入れます。

get_screenshot_on_step_start()

ステップを開始する前にスクリーンショットを作成するかどうかを返します。

get_screenshot_on_step_success()

ステップを正常に完了した後にスクリーンショットを作成するかどうかを返します。

get_screenshot_on_step_failure()

ステップが失敗した後にスクリーンショットを作成するかどうかを返します。

disable_step_screenshots()

スクリーンショットオプション (get_screenshot_on_step_start、get_screenshot_on_step_success、および get_screenshot_on_step_failure) をすべて無効にします。

enable_step_screenshots()

スクリーンショットオプション (get_screenshot_on_step_start、get_screenshot_on_step_success、および get_screenshot_on_step_failure) をすべて有効にします。デフォルトでは、これらすべてのメソッドが有効になります。

CloudWatch メトリクスに関する setConfig (オプション)

`syn-python-selenium-1.1` 以降を使用する Canary の場合、setConfig の (options) には、Canary によって発行されるメトリクスを決定する次のブール値パラメータを含めることができます。これらの各オプションのデフォルトは `true` です。`aggregated` で始まるオプションは、`CanaryName` ディメンションなしでメトリクスを出力するかどうかを決定します。これらのメトリクスを使用して、すべての Canary の集計結果を確認できます。その他のオプションは、`CanaryName` ディメンションを含むメトリクスを出力するかどうかを決定します。これらのメトリクスを使用して、個々の Canary の結果を確認できます。

Canary から発行される CloudWatch メトリクスのリストについては、「Canary によって発行される CloudWatch メトリクス」を参照してください。

- `failed_canary_metric` (ブール値) – この Canary の `Failed` メトリクス (`CanaryName` ディメンションを含む) を出力するかどうか。デフォルト: `true`。
- `failed_requests_metric` (ブール値) – この Canary の `Failed requests` メトリクス (`CanaryName` ディメンションを含む) を出力するかどうか。デフォルト: `true`。
- `2xx_metric` (ブール値) – この Canary の `2xx` メトリクス (`CanaryName` ディメンションを含む) を出力するかどうか。デフォルト: `true`。
- `4xx_metric` (ブール値) – この Canary の `4xx` メトリクス (`CanaryName` ディメンションを含む) を出力するかどうか。デフォルト: `true`。
- `5xx_metric` (ブール値) – この Canary の `5xx` メトリクス (`CanaryName` ディメンションを含む) を出力するかどうか。デフォルト: `true`。
- `step_duration_metric` (ブール値) – この Canary の `Step duration` メトリクス (`CanaryName StepName` ディメンションを含む) を出力するかどうか。デフォルト: `true`。
- `step_success_metric` (ブール値) – この Canary の `Step success` メトリクス (`CanaryName StepName` ディメンションを含む) を出力するかどうか。デフォルト: `true`。
- `aggregated_failed_canary_metric` (ブール値) – この Canary の `Failed` メトリクス (`CanaryName` ディメンションを含まない) を出力するかどうか。デフォルト: `true`。
- `aggregated_failed_requests_metric` (ブール値) – この Canary の `Failed Requests` メトリクス (`CanaryName` ディメンションを含まない) を出力するかどうか。デフォルト: `true`。

- aggregated_2xx_metric (ブール値) – この Canary の 2xx メトリクス (CanaryName ディメンションを含まない) を出力するかどうか。デフォルト: true。
- aggregated_4xx_metric (ブール値) – この Canary の 4xx メトリクス (CanaryName ディメンションを含まない) を出力するかどうか。デフォルト: true。
- aggregated_5xx_metric (ブール値) – この Canary の 5xx メトリクス (CanaryName ディメンションを含まない) を出力するかどうか。デフォルト: true。

with_2xx_metric(2xx_metric)

ブール引数を受け入れます。この引数は、この Canary に対して 2xx ディメンションを持つ CanaryName メトリクスを出力するかどうかを指定します。

with_4xx_metric(4xx_metric)

ブール引数を受け入れます。この引数は、この Canary に対して 4xx ディメンションを持つ CanaryName メトリクスを出力するかどうかを指定します。

with_5xx_metric(5xx_metric)

ブール引数を受け入れます。この引数は、この Canary に対して 5xx ディメンションを持つ CanaryName メトリクスを出力するかどうかを指定します。

withAggregated2xxMetric(aggregated2xxMetric)

ブール引数を受け入れます。この引数は、この Canary に対してディメンションを持たない 2xx メトリクスを出力するかどうかを指定します。

withAggregated4xxMetric(aggregated4xxMetric)

ブール引数を受け入れます。この引数は、この Canary に対してディメンションを持たない 4xx メトリクスを出力するかどうかを指定します。

with_aggregated_5xx_metric(aggregated_5xx_metric)

ブール引数を受け入れます。この引数は、この Canary に対してディメンションを持たない 5xx メトリクスを出力するかどうかを指定します。

with_aggregated_failed_canary_metric(aggregated_failed_canary_metric)

ブール引数を受け入れます。この引数は、この Canary に対してディメンションを持たない Failed メトリクスを出力するかどうかを指定します。

with_aggregated_failed_requests_metric(aggregated_failed_requests_metric)

ブール引数を受け入れます。この引数は、この Canary に対してディメンションを持たない `Failed requests` メトリクスを出力するかどうかを指定します。

with_failed_canary_metric(failed_canary_metric)

ブール引数を受け入れます。この引数は、この Canary に対して `Failed` ディメンションを持つ `CanaryName` メトリクスを出力するかどうかを指定します。

with_failed_requests_metric(failed_requests_metric)

ブール引数を受け入れます。この引数は、この Canary に対して `Failed requests` ディメンションを持つ `CanaryName` メトリクスを出力するかどうかを指定します。

with_step_duration_metric(step_duration_metric)

ブール引数を受け入れます。この引数は、この Canary に対して `Duration` ディメンションを持つ `CanaryName` メトリクスを出力するかどうかを指定します。

with_step_success_metric(step_success_metric)

ブール引数を受け入れます。この引数は、この Canary に対して `StepSuccess` ディメンションを持つ `CanaryName` メトリクスを出力するかどうかを指定します。

メトリクスを有効または無効にする方法

disable_aggregated_request_metrics()

Canary が `CanaryName` ディメンションなしで出力されるすべてのリクエストメトリクスを出力するのを無効にします。

disable_request_metrics()

Canary ごとのメトリクスとすべての Canary で集計されたメトリクスの両方を含む、すべてのリクエストメトリクスを無効にします。

disable_step_metrics()

ステップ成功メトリクスとステップ期間メトリクスの両方を含む、すべてのステップメトリクスを無効にします。

enable_aggregated_request_metrics()

Canary が CanaryName ディメンションなしで出力されるすべてのリクエストメトリクスを送信できるようにします。

enable_request_metrics()

Canary ごとのメトリクスとすべての Canary で集計されたメトリクスの両方を含む、すべてのリクエストメトリクスを有効にします。

enable_step_metrics()

ステップ成功メトリクスとステップ期間メトリクスの両方を含む、すべてのステップメトリクスを有効にします。

UI Canary での使用

まず、Synthetics 依存関係をインポートし、設定を取得します。次に、以下のいずれかのオプションを使用して setConfig メソッドを呼び出すことで、各オプションの設定を行います。

```
from aws_synthetics.common import synthetics_configuration

synthetics_configuration.set_config(
     {
        "screenshot_on_step_start": False,
        "screenshot_on_step_success": False,
        "screenshot_on_step_failure": True
     }
)

or
```

または

```
synthetics_configuration.with_screenshot_on_step_start(False).with_screenshot_on_step_success(F
```

すべてのスクリーンショットを無効にするには、次の例のように disableStepScreenshots() 関数を使用します。

```
synthetics_configuration.disable_step_screenshots()
```

コード内の任意の個所でスクリーンショットを有効または無効にすることができます。例えば、1 つのステップでのみスクリーンショットを無効にするには、そのステップを実行する前にスクリーンショットを無効にしてステップの実行後に有効にします。

UI Canary の set_config(options)

syn-python-selenium-1.1 から始まり、UI Canary については、set_config は次のブールパラメータを含めることができます。

- continue_on_step_failure (ブール値) – ステップが失敗した後も Canary スクリプトの実行を続行するかどうか (これは executeStep 関数を参照します)。いずれかのステップが失敗しても、Canary 実行は失敗としてマークされます。デフォルト: false。

SyntheticsLogger クラス

synthetics_logger は、コンソールと、同じログレベルのローカルログファイルの両方にログを書き込みます。このログファイルは、ログレベルが、呼び出されたログ関数の該当するログ記録レベル以下である場合にのみ、両方の場所に書き込まれます。

ローカルログファイルのログ記録ステートメントには、呼び出された関数のログレベルに合わせて「DEBUG:」や「INFO:」などが先頭に追加されます。

Amazon S3 の結果の場所にアップロードされるログファイルを作成する場合、synthetics_logger を使用する必要はありません。代わりに、別のログファイルを /tmp フォルダ内に作成できます。/tmp フォルダ内に作成したファイルは、アーティファクトとして S3 バケットの結果の場所にアップロードされます。

synthetics_logger を使用するには:

```
from aws_synthetics.common import synthetics_logger
```

有用な関数定義:

ログレベルの取得:

```
log_level = synthetics_logger.get_level()
```

ログレベルの設定:

```
synthetics_logger.set_level()
```

指定されたレベルでメッセージを口グ記録します。レベルは、次の構文の例に示すように、DEBUG、INFO、WARN、または ERROR とすることができます。

```
synthetics_logger.debug(message, *args, **kwargs)
```

```
synthetics_logger.info(message, *args, **kwargs)
```

```
synthetics_logger.log(message, *args, **kwargs)
```

```
synthetics_logger.warn(message, *args, **kwargs)
```

```
synthetics_logger.error(message, *args, **kwargs)
```

デバッグパラメータの詳細については、logging.debug の標準 Python ドキュメントをご参照ください。

これらの口ギング関数では、message はメッセージフォーマット文字列です。args は、文字列フォーマット演算子を使用して msg にマージされる引数です。

kwargs には、次の 3 つのキーワード引数があります。

- exc_info – false と評価されない場合、口グメッセージに例外情報を追加します。
- stack_info – デフォルト設定は false です。true の場合、実際の口ギング呼び出しを含むスタック情報を口ギングメッセージに追加します。
- extra – 3 番目のオプションのキーワード引数。口ギングイベント用に作成された LogRecord の __dict__ を設定するために使用されるディクショナリでユーザー定義属性を渡すために使用できます。

例:

レベル DEBUG のメッセージを口グに記録します:

```
synthetics_logger.debug('Starting step - login.')
```

レベル INFO のメッセージを口グに記録します。logger.log は logger.info のシノニムです:

```
synthetics_logger.info('Successfully completed step - login.')
```

または

```
synthetics_logger.log('Successfully completed step - login.')
```

レベル WARN のメッセージを口グに記録します:

```
synthetics_logger.warn('Warning encountered trying to publish %s', 'CloudWatch Metric')
```

レベル ERROR のメッセージを口グに記録します:

```
synthetics_logger.error('Error encountered trying to publish %s', 'CloudWatch Metric')
```

例外を口グに記録します。

```
synthetics_logger.exception(message, *args, **kwargs)
```

レベル ERROR のメッセージを口グに記録します。例外情報は、口ギングメッセージに追加されます。この関数は、例外ハンドラからのみ呼び出す必要があります。

例外パラメータの詳細については、logging.exception の標準 Python ドキュメントをご参照ください。

message はメッセージフォーマット文字列です。args は引数で、文字列フォーマット演算子を使用して msg にマージされます。

kwargs には、次の 3 つのキーワード引数があります。

- exc_info – false と評価されない場合、口グメッセージに例外情報を追加します。
- stack_info – デフォルト設定は false です。true の場合、実際の口ギング呼び出しを含むスタック情報を口ギングメッセージに追加します。
- extra – 3 番目のオプションのキーワード引数。口ギングイベント用に作成された LogRecord の __dict__ を設定するために使用されるディクショナリでユーザー定義属性を渡すために使用できます。

例:

```
synthetics_logger.exception('Error encountered trying to publish %s', 'CloudWatch
 Metric')
```

UI Canary のみに適用される Python および Selenium ライブラリクラスおよび関数

以下の Python 用の CloudWatch Synthetics Selenium ライブラリ関数は、UI Canary についてのみ有用です。

トピック

SyntheticsBrowser クラス

`synthetics_webdriver.Chrome()` を呼び出してブラウザインスタンスを作成すると、返されるブラウザインスタンスのタイプは `SyntheticsBrowser` となります。`SyntheticsBrowser` クラスは ChromeDriver を制御し、Canary スクリプトがブラウザを駆動できるように設定し、Selenium WebDriver が Synthetics で動作できるようにします。

標準的な Selenium メソッドに加えて、以下のメソッドも提供します。

set_viewport_size (幅、高さ)

ブラウザのビューポートを設定します。例:

```
browser.set_viewport_size(1920, 1080)
```

save_screenshot(ファイル名, サフィックス)

スクリーンショットを `/tmp` ディレクトリに保存します。スクリーンショットは、そこから S3 バケットの Canary アーティファクトフォルダにアップロードされます。

filename はスクリーンショットのファイル名で、suffix はスクリーンショットの命名に使用されるオプションの文字列です。

例:

```
browser.save_screenshot('loaded.png', 'page1')
```

SyntheticsWebDriver クラス

このクラスを使用するには、スクリプトで以下を使用します。

```
from aws_synthetics.selenium import synthetics_webdriver
```

add_execution_error(errorMessage, ex);

errorMessage はエラーの詳細を示し、ex は発生した例外を示します

add_execution_error を使用すると、Canary の実行エラーを設定できます。これにより、スクリプトの実行を中断することなく Canary を失敗させることができます。また、successPercent メトリクスにも影響を与えません。

Canary スクリプトの成功または失敗を確認するために重要でない場合にのみ、エラーを実行エラーとして追跡するようにします。

add_execution_error の使用例を次に示します。ここでは、エンドポイントの可用性をモニターリングしながら、ページが読み込まれた後にスクリーンショットを取得しています。スクリーンショットの作成に失敗したとしても、エンドポイントの可用性が判断されるわけではないため、スクリーンショットの作成中に発生したすべてのエラーは、キャッチした上で実行エラーとして追加します。可用性に関するメトリックスには、依然としてエンドポイントが起動し実行中であることが示されていますが、Canary のステータスは失敗としてマークされています。次に示すコードブロックのサンプルでは、このようなエラーをキャッチし、実行エラーとして追加します。

```
try:
    browser.save_screenshot("loaded.png")
except Exception as ex:
   self.add_execution_error("Unable to take screenshot", ex)
```

add_user_agent(user_agent_str)

user_agent_str の値をブラウザのユーザーエージェントヘッダーに追加します。ブラウザインスタンスを作成する前に、user_agent_str を割り当てる必要があります。

例:

```
synthetics_webdriver.add_user_agent('MyApp-1.0')
```

execute_step(step_name, function_to_execute)

1 つの関数を処理します。また、次の操作も行います。

- ステップが開始されたことをログに記録します。
- <stepName>-starting という名前でスクリーンショットを作成します。
- タイマーを開始します。
- 指定された関数を実行します。
- 関数が正常に戻ると、合格としてカウントします。関数がスローされると、失敗としてカウントします。
- タイマーを終了します。
- ステップが合格したか失敗したかをログに記録します。
- <stepName>-succeeded や <stepName>-failed という名前でスクリーンショットを作成します。
- stepName SuccessPercent メトリクスを出力します。合格の場合は 100、不合格の場合は 0 です。
- stepName Duration メトリクスを出力します。値は、ステップの開始時刻と終了時刻に基づきます。
- 最後に、functionToExecute から返された内容を返し、functionToExecute からスローされた内容を再スローします。

例:

```
from selenium.webdriver.common.by import By

def custom_actions():
        #verify contains
        browser.find_element(By.XPATH, "//*[@id=\"id_1\"][contains(text(),'login')]")
        #click a button
        browser.find_element(By.XPATH, '//*[@id="submit"]/a').click()

    await synthetics_webdriver.execute_step("verify_click", custom_actions)
```

Chrome()

Chromium ブラウザのインスタンスを起動し、作成したブラウザインスタンスを返します。

例 :

```
browser = synthetics_webdriver.Chrome()
```

```
browser.get("https://example.com/)
```

シークレットモードでブラウザを起動するには、次を使用します。

```
add_argument('—incognito')
```

プロキシ設定を追加するには、次を使用します。

```
add_argument('--proxy-server=%s' % PROXY)
```

例：

```
from selenium.webdriver.chrome.options import Options
chrome_options = Options()
chrome_options.add_argument("—incognito")
browser = syn_webdriver.Chrome(chrome_options=chrome_options)
```

cron を使用して Canary 実行をスケジュールする

cron 式を使用すると、Canary のスケジュールを柔軟に立てることができます。cron 式には、次の表に示す順序で 5 つまたは 6 つのフィールドが含まれています。フィールドはスペースで区切ります。構文は、CloudWatch コンソールを使用して Canary を作成するか、AWS CLI または AWS SDK を使用するかによって異なります。コンソールを使用する場合は、最初の 5 つのフィールドのみを指定します。AWS CLI または AWS SDK を使用する場合は、6 つのフィールドをすべて指定し、Year フィールドに * を指定する必要があります。

フィールド	許可される値	使用できる特殊文字
分	0-59	, - * /
時間	0-23	, - * /
日	1-31	, - * ? / L W
月	1-12 または JAN-DEC	, - * /
曜日	1-7 または SUN-SAT	, - * ? L #
年	*	

特殊文字

- , (カンマ) は、フィールドの式に複数の値を含めます。例えば、[月] フィールドの、「JAN,FEB,MAR」は、1 月、2 月、3 月を含みます。
- 特殊文字 - (ダッシュ) は、範囲を指定します。日フィールドの、「1-15」は、指定した月の 1 日から 15 日を含みます。
- * (アスタリスク) 特殊文字には、フィールド内のすべての値が含まれます。[時間] フィールドの * には すべての時間が含まれます。同じ式で、[日] フィールドと [曜日] フィールドの両方で * を使用することはできません。一方に使用する場合は、もう一方に [?] を使用する必要があります。
- / (スラッシュ) は増分を指定します。[分] フィールドに 1/10 と入力して、時間の最初の分から開始して 10 分おきを指定できます (例えば、11 分、21 分、31 分など)。
- ? (疑問符) は、任意を意味します。[日] フィールドに 7 と入力し、7 番目の曜日がどの曜日か気にしない場合は、?を [曜日] フィールドに入力します。
- Day-of-month フィールドまたは Day-of-week フィールドの、ワイルドカード L は月または週の最終日を指定します。
- Day-of-month フィールドのワイルドカード **W** は、平日を指定します。Day-of-month フィールドで、**3W** は月の 3 日目に最も近い平日を指定します。
- Day-of-week フィールドの # ワイルドカードは、月の指定された曜日の特定のインスタンスを指定します。例えば、3#2 は月の第 2 火曜日です。3 は毎週の 3 日目であるため、火曜日を指し、2 は月内のその種類の 2 つ目を指します。

制限事項

- cron 式の日フィールドと曜日フィールドを同時に指定することはできません。一方のフィールドに値または * (アスタリスク) を指定する場合、もう一方のフィールドで ? (疑問符) を使用する必要があります。
- 1 分より短い間隔を導き出す cron 式はサポートされていません。
- 実行前に 1 年以上待つように Canary を設定することはできないため、[Year] フィールドでは * のみ指定することができます。

例

Canary を作成するときは、次のサンプルの cron 文字列を参照できます。以下の例は、AWS CLI または AWS SDK を使用して Canary を作成または更新するための正しい構文です。CloudWatch コンソールを使用している場合は、各例の最後の 「*」を省略します。

式	意味
`0 10 * * ? *`	毎日午前 10:00 (UTC) に実行
`15 12 * * ? *`	毎日午後 12:15 (UTC) に実行
`0 18 ? * MON-FRI *`	毎週月曜日から金曜日まで午後 6:00 (UTC) に実行
`0 8 1 * ? *`	毎月初日の午前 8:00 (UTC) に実行
`0/10 * ? * MON-SAT *`	毎週月曜日から土曜日まで 10 分ごとに実行
`0/5 8-17 ? * MON-FRI *`	月曜日から金曜日まで午前 8:00 から午後 5:55 (UTC) まで 5 分ごとに実行

グループ

クロスリージョンの canary を含む、相互に canary を関連付けるグループを作成することができます。グループを使用することで、canary の管理と自動化に役立ちます。また、グループ内のすべての canary の集計された実行結果と統計を表示することもできます。

グループはグローバルリソースです。グループを作成すると、グループをサポートするすべての AWS リージョンに渡って複製されます。これらのリージョンのいずれかから canary を追加することができ、これらのリージョンのすべてで表示できるようになります。グループ ARN 形式には作成されたリージョン名が反映されますが、グループはリージョンに制約されません。これは、複数のリージョンからの canary を同じグループに配置し、そのグループを使用して、それらすべての canary を単一のビューで表示および管理できることを意味します。

グループは、デフォルトで無効になっているリージョンを除くすべてのリージョンでサポートされています。これらのリージョンの詳細については、「Enabling a Region」(リージョンを有効にする) を参照してください。

各グループには最大 10 個の canary を含めることができます。アカウントには最大 20 個のグループを含めることができます。単一の canary はすべて、最大 10 個のグループのメンバーになることができます。

グループを作成するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。

2. ナビゲーションペインで、[Application Signals]、[Synthetics Canary] の順に選択します。
3. グループを作成 を選択します。
4. [Group Name] (グループ名) で、グループ名を入力します。
5. このグループに関連付いた canary を選択します。canary を選択するには、その完全な名前を[Exact canary name] (完全な canary 名) に入力し、[Search] (検索) を選択します。その後、canary 名の横にあるチェックボックスを選択します。異なるリージョンに同じ名前の canary が複数ある場合は、必要な canary を必ず選択してください。

   この手順を繰り返して、最大 10 個の canary をグループに関連付けることができます。
6. (オプション) [Tags] (タグ) で、このグループのタグとする 1 つ以上のキーと値のペアを追加します。タグを使用すると、AWS リソースを識別して整理したり、AWS コストを追跡したりしやすくなります。詳細については、Amazon CloudWatch リソースにタグを付ける を参照してください。
7. グループを作成 を選択します。

Canary をローカルでテストする

このセクションでは、Microsoft Visual Studio コードエディタまたは JetBrains IDE コードエディタ内で CloudWatch Synthetics Canary を直接変更、テスト、デバッグする方法について解説します。ローカルデバッグ環境では、サーバーレスアプリケーションモデル (SAM) コンテナを使用して Lambda 関数をシミュレートし、Synthetics Canary の動作をエミュレートします。

> **Note**
>
> ビジュアルモニタリングに依存する Canary をローカルでデバッグすることは現実的ではありません。ビジュアルモニタリングは、初回実行時にベーススクリーンショットをキャプチャし、その後、これらのスクリーンショットを後続の実行のスクリーンショットと比較します。ローカルの開発環境では実行は保存または追跡されず、反復処理はその都度独立してスタンドアロンで実行されます。Canary の実行履歴がないため、ビジュアルモニタリングに依存する Canary をデバッグすることは現実的ではありません。

前提条件

1. HAR ファイルやスクリーンショットなど、ローカルの Canary テストランのアーティファクトを保存するための Amazon S3 バケットを選択または作成します。そのためには、IAM でプロビジョ

ニングする必要があります。Amazon S3 バケットの設定をスキップしても Canary をローカルでテストすることはできますが、欠落しているバケットに関するエラーメッセージが表示されて Canary アーティファクトにアクセスできなくなります。

Amazon S3 バケットを使用する場合は、数日後にオブジェクトを削除するようにバケットのライフサイクルを設定すると、コストを節約できるためお勧めです。詳細については、「Managing your storage lifecycle」を参照してください。

2. AWS アカウントのデフォルトの AWS プロファイルを設定します。詳細については、「Configuration and credential file settings」を参照してください。
3. デバッグ環境のデフォルトの AWS リージョンを、`us-west-2` など任意のリージョンに設定します。
4. AWS SAM CLI をインストールします。詳細については、「AWS SAM CLI のインストール」を参照してください。
5. Visual Studio Code Editor または JetBrains IDE をインストールします。詳細については、「Visual Studio Code」または「JetBrains IDE」を参照してください。
6. AWS SAM CLI と連携させるため Docker をインストールします。必ず docker デーモンを起動します。詳細については、「Installing Docker to use with the AWS SAM CLI」を参照してください。

   あるいは、Docker ランタイムを使用している間は Rancher など他のコンテナ管理ソフトウェアをインストールすることもできます。
7. 任意のエディタに AWS ツールキット拡張機能をインストールします。詳細については、「Installing the AWS Toolkit for Visual Studio Code」または「Installing the AWS Toolkit for JetBrains」を参照してください。

トピック

テストおよびデバッグ環境のセットアップ

まず、次のコマンドを入力して、AWS が提供する Github リポジトリのクローンを作成します。このリポジトリには、Node.js Canary と Python Canary 両方のコードサンプルが含まれています。

```
git clone https://github.com/aws-samples/synthetics-canary-local-debugging-sample.git
```

次に、Canary の言語に応じて、次のいずれかを実行します。

Node.js Canary の場合

1. 次のコマンドを入力して、Node.js Canary ソースディレクトリに移動します。

   ```
   cd synthetics-canary-local-debugging-sample/nodejs-canary/src
   ```

2. 次のコマンドを入力して、Canary の依存関係をインストールします。

   ```
   npm install
   ```

Python Canary の場合

1. 次のコマンドを入力して、Python Canary ソースディレクトリに移動します。

   ```
   cd synthetics-canary-local-debugging-sample/python-canary/src
   ```

2. 次のコマンドを入力して、Canary の依存関係をインストールします。

   ```
   pip3 install -r requirements.txt -t .
   ```

Visual Studio Code IDE を使用します。

Visual Studio 起動設定ファイルは、`.vscode/launch.json` にあります。これには、テンプレートファイルを Visual Studio コードで検出できるようにする設定が含まれています。Lambda ペイロードを、Canary の呼び出しに必要なパラメータを含めて定義します。Node.js Canary の起動設定は次のとおりです。

```
{
            ...
            ...
```

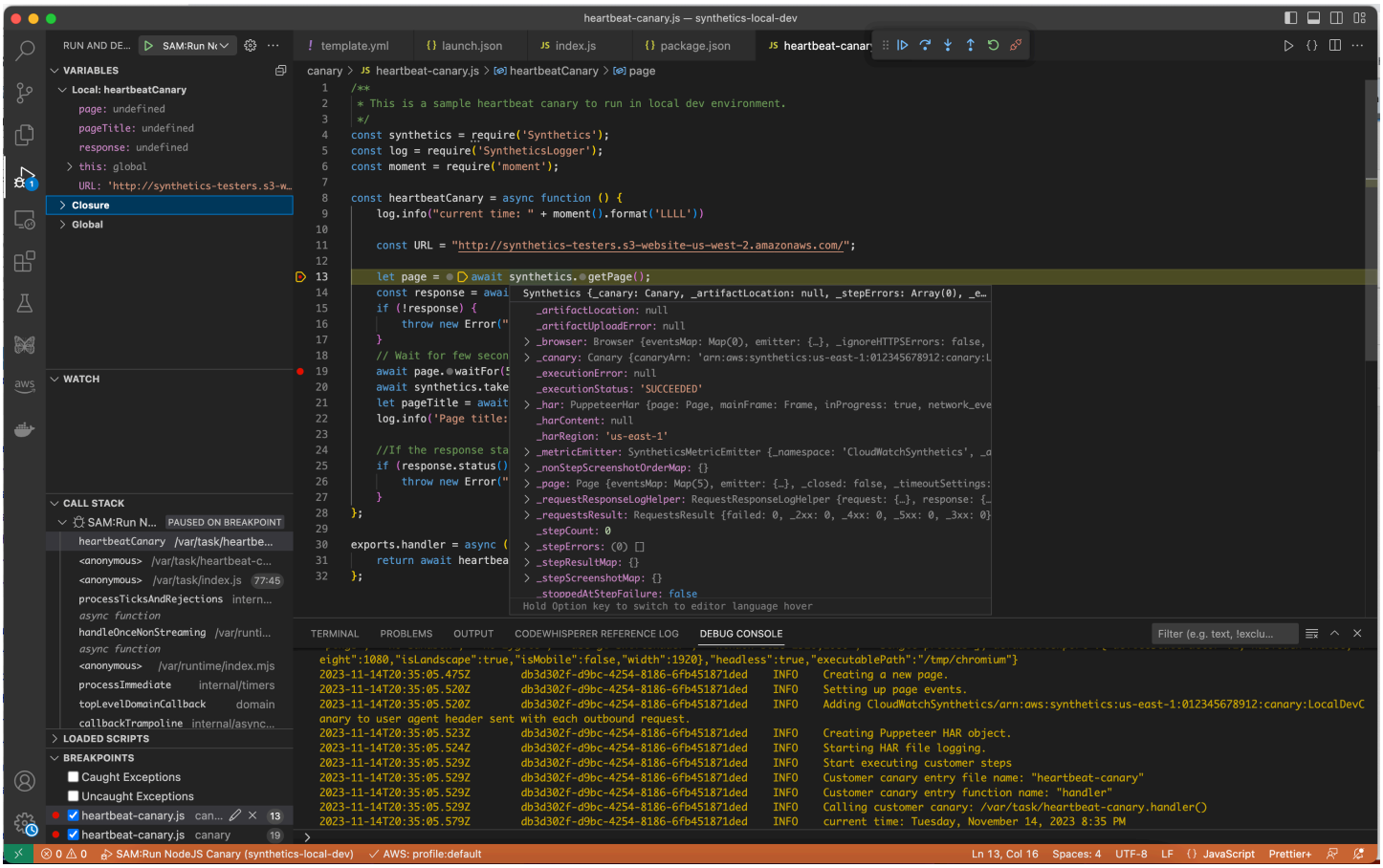
```
            "lambda": {
                "payload": {
                    "json": {
                        // Canary name. Provide any name you like.
                        "canaryName": "LocalSyntheticsCanary",
                        // Canary artifact location
                        "artifactS3Location": {
                            "s3Bucket": "cw-syn-results-123456789012-us-west-2",
                            "s3Key": "local-run-artifacts",
                        },
                        // Your canary handler name
                        "customerCanaryHandlerName": "heartbeat-canary.handler"
                    }
                },
                // Environment variables to pass to the canary code
                "environmentVariables": {}
            }
        }
    ]
}
```

オプションで、ペイロード JSON に次のフィールドを指定することもできます。

- `s3EncryptionMode` 有効な値: `SSE_S3` | `SSE_KMS`
- `s3KmsKeyArn` 有効な値: *`KMS ## ARN`*
- `activeTracing` 有効な値: `true` | `false`
- `canaryRunId` 有効な値: *`UUID`* このパラメータは、アクティブトレースが有効になっている場合に必要になります。

Visual Studio で Canary をデバッグするには、実行を一時停止する Canary コードにブレークポイントを追加します。ブレークポイントを追加するには、エディタの余白を選択し、エディタの [実行とデバッグ] モードに移動します。[再生] をクリックして Canary を実行します。Canary が実行されると、ログがデバッグコンソールで追跡され、Canary の動作に関するリアルタイムインサイトが表示されます。ブレークポイントを追加すると、Canary の実行は各ブレークポイントで一時停止するため、コードをステップスルーして変数値、インスタンスメソッド、オブジェクト属性、関数呼び出しスタックをチェックすることができます。

Canary をローカルで実行およびデバッグするのにコストはかかりません (Amazon S3 バケットに保存されているアーティファクトと、各ローカル実行によって生成された CloudWatch メトリクスは除きます)。



JetBrains IDE を使用します。

AWS Toolkit for JetBrains 拡張機能をインストールした後は、Node.js Canary をデバッグする場合、Node.js プラグインと JavaScript デバッガの実行が有効になっていることを確認します。次に、以下の手順に従います。

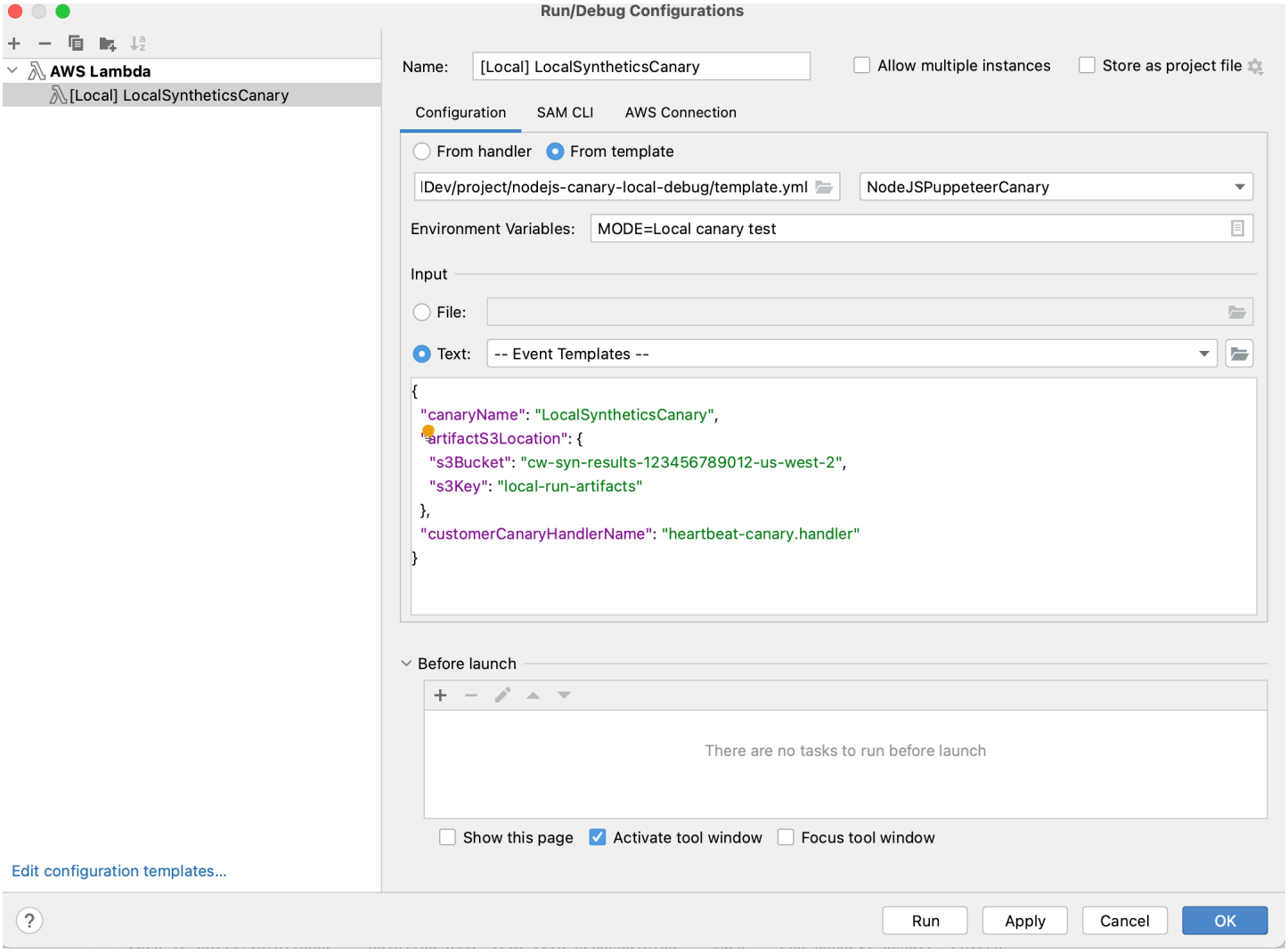
JetBrains IDE を使用して Canary をデバッグする

1. JetBrains IDE の左側のナビゲーションペインで [Lambda] を選択し、次に、ローカルの設定テンプレートを選択します。
2. 実行設定の名前を入力します (**`LocalSyntheticsCanary`** など)。
3. [テンプレートから] をクリックし、テンプレートフィールドでファイルブラウザを選択して、プロジェクトの nodejs ディレクトリまたは python ディレクトリから template.yml ファイルを選択してます。
4. [入力] セクションで、以下に示すように Canary のペイロードを入力します。

```
{
```

```
 "canaryName": "LocalSyntheticsCanary",
 "artifactS3Location": {
     "s3Bucket": "cw-syn-results-123456789012-us-west-2",
     "s3Key": "local-run-artifacts"
 },
 "customerCanaryHandlerName": "heartbeat-canary.handler"
}
```

Visual Studio Code IDE を使用します。 に示すように、他の環境変数をペイロード JSON で設定することもできます。



SAM CLI を使用して Canary をローカルで実行する

サーバーレスアプリケーションモデル (AWS SAM) CLI を使用して Canary をローカルで実行するには、次のいずれかの手順を使用します。`event.json` で、`s3Bucket` に独自の Amazon S3 バケット名を指定します。

SAM CLI を使用して Node.js Canary を実行するには

1. 次のコマンドを入力してソースディレクトリに移動します。

```
cd synthetics-canary-local-debugging-sample/nodejs-canary
```

2. 次のコマンドを入力します。

```
sam build
sam local invoke -e ../event.json
```

SAM CLI を使用して Python Canary を実行するには

1. 次のコマンドを入力してソースディレクトリに移動します。

```
cd synthetics-canary-local-debugging-sample/python-canary
```

2. 次のコマンドを入力します。

```
sam build
sam local invoke -e ../event.json
```

ローカルのテスト環境を既存の Canary パッケージに統合する

ローカルの Canary デバッグは、次の 3 つのファイルをコピーすることで既存の Canary パッケージに統合することができます。

- `template.yml` ファイルを Canary パッケージルートにコピーします。`CodeUri` のパスを、Canary コードがあるディレクトリを指すように変更します。
- Node.js Canary を使用している場合は、`cw-synthetics.js` ファイルを Canary ソースディレクトリにコピーします。Python Canary を使用している場合は、`cw-synthetics.py` ファイルを Canary ソースディレクトリにコピーします。

- 起動設定ファイル .vscode/launch.json をパッケージルートにコピーします。これが .vscode ディレクトリ内にあることを確認し、ない場合は作成します。

CloudWatch Synthetics ランタイムを変更する

デバッグの一環として、最新のランタイムではなく、別の CloudWatch Synthetics ランタイムを使って Canary を実行します。これを行うには、次の表の中から使用するランタイムを見つけます。必ず正しいリージョンのランタイムを選択します。次に、そのランタイムの ARN を template.yml ファイル内の適切な場所に貼り付け、Canary を実行します。

Node.js ランタイム

ARNs for syn-nodejs-puppeteer-7.0

次の表は、利用可能な各 AWS リージョンで、CloudWatch Synthetics ランタイムのバージョン syn-nodejs-puppeteer-7.0 に使用する ARN の一覧です。

リージョン	ARN
米国東部 (バージニア北部)	arn:aws:lambda:us-east-1:378653112637:layer:Synthetics:44
米国東部 (オハイオ)	arn:aws:lambda:us-east-2:772927465453:layer:Synthetics:46
米国西部 (北カリフォルニア)	arn:aws:lambda:us-west-1:332033056316:layer:Synthetics:44
米国西部 (オレゴン)	arn:aws:lambda:us-west-2:760325925879:layer:Synthetics:47
アフリカ (ケープタウン)	arn:aws:lambda:af-south-1:461844272066:layer:Synthetics:44
アジアパシフィック (香港)	arn:aws:lambda:ap-east-1:129828061636:layer:Synthetics:45
アジアパシフィック (ハイデラバード)	arn:aws:lambda:ap-south-2:280298676434:layer:Synthetics:20

リージョン	ARN
アジアパシフィック (ジャカルタ)	arn:aws:lambda:ap-southeast-3:246953257743:layer:Synthetics:26
アジアパシフィック (メルボルン)	arn:aws:lambda:ap-southeast-4:200724813040:layer:Synthetics:18
アジアパシフィック (ムンバイ)	arn:aws:lambda:ap-south-1:724929286329:layer:Synthetics:44
アジアパシフィック (大阪)	arn:aws:lambda:ap-northeast-3:608016332111:layer:Synthetics:30
アジアパシフィック (ソウル)	arn:aws:lambda:ap-northeast-2:989515803484:layer:Synthetics:46
アジアパシフィック (シンガポール)	arn:aws:lambda:ap-southeast-1:068035103298:layer:Synthetics:49
アジアパシフィック (シドニー)	arn:aws:lambda:ap-southeast-2:584677157514:layer:Synthetics:44
アジアパシフィック (東京)	arn:aws:lambda:ap-northeast-1:172291836251:layer:Synthetics:44
カナダ (中部)	arn:aws:lambda:ca-central-1:236629016841:layer:Synthetics:44
カナダ西部 (カルガリー)	arn:aws:lambda:ca-west-1:944448206667:layer:Synthetics:76
中国 (北京)	arn:aws-cn:lambda:cn-north-1:422629156088:layer:Synthetics:45
中国 (寧夏)	arn:aws-cn:lambda:cn-northwest-1:474974519687:layer:Synthetics:46
欧州 (フランクフルト)	arn:aws:lambda:eu-central-1:122305336817:layer:Synthetics:44

リージョン	ARN
欧州 (アイルランド)	arn:aws:lambda:eu-west-1:563204233543:layer:Synthetics:46
欧州 (ロンドン)	arn:aws:lambda:eu-west-2:565831452869:layer:Synthetics:44
欧州 (ミラノ)	arn:aws:lambda:eu-south-1:525618516618:layer:Synthetics:45
欧州 (パリ)	arn:aws:lambda:eu-west-3:469466506258:layer:Synthetics:44
欧州 (スペイン)	arn:aws:lambda:eu-south-2:029793053121:layer:Synthetics:20
欧州 (ストックホルム)	arn:aws:lambda:eu-north-1:162938142733:layer:Synthetics:44
欧州 (チューリッヒ)	arn:aws:lambda:eu-central-2:224218992030:layer:Synthetics:19
イスラエル (テルアビブ)	arn:aws:lambda:il-central-1:313249807427:layer:Synthetics:17
中東 (バーレーン)	arn:aws:lambda:me-south-1:823195537320:layer:Synthetics:44
中東 (アラブ首長国連邦)	arn:aws:lambda:me-central-1:239544149032:layer:Synthetics:19
南米 (サンパウロ)	arn:aws:lambda:sa-east-1:783765544751:layer:Synthetics:45
AWS GovCloud (米国東部)	arn:aws-us-gov:lambda:us-gov-east-1:946759330430:layer:Synthetics:41
AWS GovCloud (米国西部)	arn:aws-us-gov:lambda:us-gov-west-1:946807836238:layer:Synthetics:42

ARNs for syn-nodejs-puppeteer-6.2

次の表は、利用可能な各 AWS リージョンで、CloudWatch Synthetics ランタイムのバージョン syn-nodejs-puppeteer-6.2 に使用する ARN の一覧です。

リージョン	ARN
米国東部 (バージニア北部)	arn:aws:lambda:us-east-1:378653112637:layer:Synthetics:41
米国東部 (オハイオ)	arn:aws:lambda:us-east-2:772927465453:layer:Synthetics:43
米国西部 (北カリフォルニア)	arn:aws:lambda:us-west-1:332033056316:layer:Synthetics:41
米国西部 (オレゴン)	arn:aws:lambda:us-west-2:760325925879:layer:Synthetics:44
アフリカ (ケープタウン)	arn:aws:lambda:af-south-1:461844272066:layer:Synthetics:41
アジアパシフィック (香港)	arn:aws:lambda:ap-east-1:129828061636:layer:Synthetics:42
アジアパシフィック (ハイデラバード)	arn:aws:lambda:ap-south-2:280298676434:layer:Synthetics:17
アジアパシフィック (ジャカルタ)	arn:aws:lambda:ap-southeast-3:246953257743:layer:Synthetics:23
アジアパシフィック (メルボルン)	arn:aws:lambda:ap-southeast-4:200724813040:layer:Synthetics:15
アジアパシフィック (ムンバイ)	arn:aws:lambda:ap-south-1:724929286329:layer:Synthetics:41
アジアパシフィック (大阪)	arn:aws:lambda:ap-northeast-3:608016332111:layer:Synthetics:27

リージョン	ARN
アジアパシフィック (ソウル)	arn:aws:lambda:ap-northeast-2:989515803484:layer:Synthetics:42
アジアパシフィック (シンガポール)	arn:aws:lambda:ap-southeast-1:068035103298:layer:Synthetics:46
アジアパシフィック (シドニー)	arn:aws:lambda:ap-southeast-2:584677157514:layer:Synthetics:41
アジアパシフィック (東京)	arn:aws:lambda:ap-northeast-1:172291836251:layer:Synthetics:41
カナダ (中部)	arn:aws:lambda:ca-central-1:236629016841:layer:Synthetics:41
カナダ西部 (カルガリー)	arn:aws:lambda:ca-west-1:944448206667:layer:Synthetics:73
中国 (北京)	arn:aws-cn:lambda:cn-north-1:422629156088:layer:Synthetics:42
中国 (寧夏)	arn:aws-cn:lambda:cn-northwest-1:474974519687:layer:Synthetics:43
欧州 (フランクフルト)	arn:aws:lambda:eu-central-1:122305336817:layer:Synthetics:41
欧州 (アイルランド)	arn:aws:lambda:eu-west-1:563204233543:layer:Synthetics:43
欧州 (ロンドン)	arn:aws:lambda:eu-west-2:565831452869:layer:Synthetics:41
欧州 (ミラノ)	arn:aws:lambda:eu-south-1:525618516618:layer:Synthetics:42
欧州 (パリ)	arn:aws:lambda:eu-west-3:469466506258:layer:Synthetics:41

リージョン	ARN
欧州 (スペイン)	arn:aws:lambda:eu-south-2:029793053121:layer:Synthetics:17
欧州 (ストックホルム)	arn:aws:lambda:eu-north-1:162938142733:layer:Synthetics:41
欧州 (チューリッヒ)	arn:aws:lambda:eu-central-2:224218992030:layer:Synthetics:16
イスラエル (テルアビブ)	arn:aws:lambda:il-central-1:313249807427:layer:Synthetics:14
中東 (バーレーン)	arn:aws:lambda:me-south-1:823195537320:layer:Synthetics:41
中東 (アラブ首長国連邦)	arn:aws:lambda:me-central-1:239544149032:layer:Synthetics:16
南米 (サンパウロ)	arn:aws:lambda:sa-east-1:783765544751:layer:Synthetics:42
AWS GovCloud (米国東部)	arn:aws-us-gov:lambda:us-gov-east-1:946759330430:layer:Synthetics:39
AWS GovCloud (米国西部)	arn:aws-us-gov:lambda:us-gov-west-1:946807836238:layer:Synthetics:39

ARNs for syn-nodejs-puppeteer-5.2

次の表は、利用可能な各 AWS リージョンで、CloudWatch Synthetics ランタイムのバージョン `syn-nodejs-puppeteer-5.2` に使用する ARN の一覧です。

リージョン	ARN
米国東部 (バージニア北部)	arn:aws:lambda:us-east-1:378653112637:layer:Synthetics:42

リージョン	ARN
米国東部 (オハイオ)	arn:aws:lambda:us-east-2:772927465453:layer:Synthetics:44
米国西部 (北カリフォルニア)	arn:aws:lambda:us-west-1:332033056316:layer:Synthetics:42
米国西部 (オレゴン)	arn:aws:lambda:us-west-2:760325925879:layer:Synthetics:45
アフリカ (ケープタウン)	arn:aws:lambda:af-south-1:461844272066:layer:Synthetics:42
アジアパシフィック (香港)	arn:aws:lambda:ap-east-1:129828061636:layer:Synthetics:43
アジアパシフィック (ハイデラバード)	arn:aws:lambda:ap-south-2:280298676434:layer:Synthetics:18
アジアパシフィック (ジャカルタ)	arn:aws:lambda:ap-southeast-3:246953257743:layer:Synthetics:24
アジアパシフィック (メルボルン)	arn:aws:lambda:ap-southeast-4:200724813040:layer:Synthetics:16
アジアパシフィック (ムンバイ)	arn:aws:lambda:ap-south-1:724929286329:layer:Synthetics:42
アジアパシフィック (大阪)	arn:aws:lambda:ap-northeast-3:608016332111:layer:Synthetics:28
アジアパシフィック (ソウル)	arn:aws:lambda:ap-northeast-2:989515803484:layer:Synthetics:44
アジアパシフィック (シンガポール)	arn:aws:lambda:ap-southeast-1:068035103298:layer:Synthetics:47
アジアパシフィック (シドニー)	arn:aws:lambda:ap-southeast-2:584677157514:layer:Synthetics:42

リージョン	ARN
アジアパシフィック (東京)	arn:aws:lambda:ap-northeast-1:172291836251:layer:Synthetics:42
カナダ (中部)	arn:aws:lambda:ca-central-1:236629016841:layer:Synthetics:42
カナダ西部 (カルガリー)	arn:aws:lambda:ca-west-1:944448206667:layer:Synthetics:74
中国 (北京)	arn:aws-cn:lambda:cn-north-1:422629156088:layer:Synthetics:43
中国 (寧夏)	arn:aws-cn:lambda:cn-northwest-1:474974519687:layer:Synthetics:44
欧州 (フランクフルト)	arn:aws:lambda:eu-central-1:122305336817:layer:Synthetics:42
欧州 (アイルランド)	arn:aws:lambda:eu-west-1:563204233543:layer:Synthetics:44
欧州 (ロンドン)	arn:aws:lambda:eu-west-2:565831452869:layer:Synthetics:42
欧州 (ミラノ)	arn:aws:lambda:eu-south-1:525618516618:layer:Synthetics:43
欧州 (パリ)	arn:aws:lambda:eu-west-3:469466506258:layer:Synthetics:42
欧州 (スペイン)	arn:aws:lambda:eu-south-2:029793053121:layer:Synthetics:18
欧州 (ストックホルム)	arn:aws:lambda:eu-north-1:162938142733:layer:Synthetics:42
欧州 (チューリッヒ)	arn:aws:lambda:eu-central-2:224218992030:layer:Synthetics:17

リージョン	ARN
イスラエル (テルアビブ)	arn:aws:lambda:il-central-1:313249807427:layer:Synthetics:15
中東 (バーレーン)	arn:aws:lambda:me-south-1:823195537320:layer:Synthetics:42
中東 (アラブ首長国連邦)	arn:aws:lambda:me-central-1:239544149032:layer:Synthetics:17
南米 (サンパウロ)	arn:aws:lambda:sa-east-1:783765544751:layer:Synthetics:43
AWS GovCloud (米国東部)	arn:aws-us-gov:lambda:us-gov-east-1:946759330430:layer:Synthetics:40
AWS GovCloud (米国西部)	arn:aws-us-gov:lambda:us-gov-west-1:946807836238:layer:Synthetics:40

Python ランタイム

ARNs for syn-python-selenium-3.0

次の表は、利用可能な各 AWS リージョンで、CloudWatch Synthetics ランタイムのバージョン `syn-python-selenium-3.0` に使用する ARN の一覧です。

リージョン	ARN
米国東部 (バージニア北部)	aarn:aws:lambda:us-east-1:378653112637:layer:Synthetics_Selenium:32
米国東部 (オハイオ)	arn:aws:lambda:us-east-2:772927465453:layer:Synthetics_Selenium:34
米国西部 (北カリフォルニア)	arn:aws:lambda:us-west-1:332033056316:layer:Synthetics_Selenium:32

リージョン	ARN
米国西部 (オレゴン)	arn:aws:lambda:us-west-2:760325925879:layer:Synthetics_Selenium:34
アフリカ (ケープタウン)	arn:aws:lambda:af-south-1:461844272066:layer:Synthetics_Selenium:32
アジアパシフィック (香港)	arn:aws:lambda:ap-east-1:129828061636:layer:Synthetics_Selenium:32
アジアパシフィック (ハイデラバード)	arn:aws:lambda:ap-south-2:280298676434:layer:Synthetics_Selenium:20
アジアパシフィック (ジャカルタ)	arn:aws:lambda:ap-southeast-3:246953257743:layer:Synthetics_Selenium:26
アジアパシフィック (メルボルン)	arn:aws:lambda:ap-southeast-4:200724813040:layer:Synthetics_Selenium:18
アジアパシフィック (ムンバイ)	arn:aws:lambda:ap-south-1:724929286329:layer:Synthetics_Selenium:32
アジアパシフィック (大阪)	arn:aws:lambda:ap-northeast-3:608016332111:layer:Synthetics_Selenium:30
アジアパシフィック (ソウル)	arn:aws:lambda:ap-northeast-2:989515803484:layer:Synthetics_Selenium:34
アジアパシフィック (シンガポール)	arn:aws:lambda:ap-southeast-1:068035103298:layer:Synthetics_Selenium:37
アジアパシフィック (シドニー)	arn:aws:lambda:ap-southeast-2:584677157514:layer:Synthetics_Selenium:32
アジアパシフィック (東京)	arn:aws:lambda:ap-northeast-1:172291836251:layer:Synthetics_Selenium:32
カナダ (中部)	arn:aws:lambda:ca-central-1:236629016841:layer:Synthetics_Selenium:32

リージョン	ARN
カナダ西部 (カルガリー)	arn:aws:lambda:ca-west-1:944448206667:layer:Synthetics_Selenium:76
中国 (北京)	arn:aws-cn:lambda:cn-north-1:422629156088:layer:Synthetics_Selenium:32
中国 (寧夏)	arn:aws-cn:lambda:cn-northwest-1:474974519687:layer:Synthetics_Selenium:32
欧州 (フランクフルト)	arn:aws:lambda:eu-central-1:122305336817:layer:Synthetics_Selenium:32
欧州 (アイルランド)	arn:aws:lambda:eu-west-1:563204233543:layer:Synthetics_Selenium:34
欧州 (ロンドン)	arn:aws:lambda:eu-west-2:565831452869:layer:Synthetics_Selenium:32
欧州 (ミラノ)	arn:aws:lambda:eu-south-1:525618516618:layer:Synthetics_Selenium:33
欧州 (パリ)	arn:aws:lambda:eu-west-3:469466506258:layer:Synthetics_Selenium:32
欧州 (スペイン)	arn:aws:lambda:eu-south-2:029793053121:layer:Synthetics_Selenium:20
欧州 (ストックホルム)	arn:aws:lambda:eu-north-1:162938142733:layer:Synthetics_Selenium:32
欧州 (チューリッヒ)	arn:aws:lambda:eu-central-2:224218992030:layer:Synthetics_Selenium:19
イスラエル (テルアビブ)	arn:aws:lambda:il-central-1:313249807427:layer:Synthetics_Selenium:17
中東 (バーレーン)	arn:aws:lambda:me-south-1:823195537320:layer:Synthetics_Selenium:32

リージョン	ARN
中東 (アラブ首長国連邦)	arn:aws:lambda:me-central-1:239544149032:layer:Synthetics_Selenium:19
南米 (サンパウロ)	arn:aws:lambda:sa-east-1:783765544751:layer:Synthetics_Selenium:33
AWS GovCloud (米国東部)	arn:aws-us-gov:lambda:us-gov-east-1:946759330430:layer:Synthetics_Selenium:30
AWS GovCloud (米国西部)	arn:aws-us-gov:lambda:us-gov-west-1:946807836238:layer:Synthetics_Selenium:31

ARNs for syn-python-selenium-2.1

次の表は、利用可能な各 AWS リージョンで、CloudWatch Synthetics ランタイムのバージョン syn-python-selenium-2.1 に使用する ARN の一覧です。

リージョン	ARN
米国東部 (バージニア北部)	arn:aws:lambda:us-east-1:378653112637:layer:Synthetics:29
米国東部 (オハイオ)	arn:aws:lambda:us-east-2:772927465453:layer:Synthetics:31
米国西部 (北カリフォルニア)	arn:aws:lambda:us-west-1:332033056316:layer:Synthetics:29
米国西部 (オレゴン)	arn:aws:lambda:us-west-2:760325925879:layer:Synthetics:31
アフリカ (ケープタウン)	arn:aws:lambda:af-south-1:461844272066:layer:Synthetics:29
アジアパシフィック (香港)	arn:aws:lambda:ap-east-1:129828061636:layer:Synthetics:29

リージョン	ARN
アジアパシフィック (ハイデラバード)	arn:aws:lambda:ap-south-2:280298676434:layer:Synthetics:17
アジアパシフィック (ジャカルタ)	arn:aws:lambda:ap-southeast-3:246953257743:layer:Synthetics:23
アジアパシフィック (メルボルン)	arn:aws:lambda:ap-southeast-4:200724813040:layer:Synthetics:15
アジアパシフィック (ムンバイ)	arn:aws:lambda:ap-south-1:724929286329:layer:Synthetics:29
アジアパシフィック (大阪)	arn:aws:lambda:ap-northeast-3:608016332111:layer:Synthetics:27
アジアパシフィック (ソウル)	arn:aws:lambda:ap-northeast-2:989515803484:layer:Synthetics:30
アジアパシフィック (シンガポール)	arn:aws:lambda:ap-southeast-1:068035103298:layer:Synthetics:34
アジアパシフィック (シドニー)	arn:aws:lambda:ap-southeast-2:584677157514:layer:Synthetics:29
アジアパシフィック (東京)	arn:aws:lambda:ap-northeast-1:172291836251:layer:Synthetics:29
カナダ (中部)	arn:aws:lambda:ca-central-1:236629016841:layer:Synthetics:29
カナダ西部 (カルガリー)	arn:aws:lambda:ca-west-1:944448206667:layer:Synthetics:73
中国 (北京)	arn:aws-cn:lambda:cn-north-1:422629156088:layer:Synthetics:29
中国 (寧夏)	arn:aws-cn:lambda:cn-northwest-1:474974519687:layer:Synthetics:29

リージョン	ARN
欧州 (フランクフルト)	arn:aws:lambda:eu-central-1:122305336817:layer:Synthetics:29
欧州 (アイルランド)	arn:aws:lambda:eu-west-1:563204233543:layer:Synthetics:31
欧州 (ロンドン)	arn:aws:lambda:eu-west-2:565831452869:layer:Synthetics:29
欧州 (ミラノ)	arn:aws:lambda:eu-south-1:525618516618:layer:Synthetics:30
欧州 (パリ)	arn:aws:lambda:eu-west-3:469466506258:layer:Synthetics:29
欧州 (スペイン)	arn:aws:lambda:eu-south-2:029793053121:layer:Synthetics:17
欧州 (ストックホルム)	arn:aws:lambda:eu-north-1:162938142733:layer:Synthetics:29
欧州 (チューリッヒ)	arn:aws:lambda:eu-central-2:224218992030:layer:Synthetics:16
イスラエル (テルアビブ)	arn:aws:lambda:il-central-1:313249807427:layer:Synthetics:14
中東 (バーレーン)	arn:aws:lambda:me-south-1:823195537320:layer:Synthetics:29
中東 (アラブ首長国連邦)	arn:aws:lambda:me-central-1:239544149032:layer:Synthetics:16
南米 (サンパウロ)	arn:aws:lambda:sa-east-1:783765544751:layer:Synthetics:30
AWS GovCloud (米国東部)	arn:aws-us-gov:lambda:us-gov-east-1:946759330430:layer:Synthetics:29

リージョン	ARN
AWS GovCloud (米国西部)	`arn:aws-us-gov:lambda:us-gov-west-1:946807836238:layer:Synthetics:29`

一般的なエラー

エラー: AWS SAM プロジェクトをローカルで実行するときは Docker が必要です。Docker をインストールし、実行していますか。

お使いのコンピュータで Docker を起動します。

SAM ローカル呼び出しに失敗しました: GetLayerVersion オペレーションを呼び出すときにエラー (ExpiredTokenException) が発生しました: リクエストに含まれるセキュリティトークンの有効期限が切れています

AWS デフォルトプロファイルが設定されていることを確認します。

よくあるエラー

SAM のよくあるエラーについての詳細は、「AWS SAM CLI troubleshooting 」を参照してください。

失敗した Canary のトラブルシューティング

Canary が失敗した場合は、以下を確認しトラブルシューティングしてください。

一般的なトラブルシューティング

- Canary の詳細ページを開き、詳細を確認します。CloudWatch コンソールのナビゲーションウィンドウで [Canaries] をクリックし、詳細ページを開きたい Canary の名前を選択します。[可用性] タブで [SuccessPercent] メトリクスを参照し、発生している問題が持続的か断続的かを確認します。

  同じ [可用性] タブで、失敗したデータポイントを選択することで、失敗した実行のスクリーンショット、ログ、および (それがある場合には) ステップレポートを見ることができます。

  ステップがスクリプトの一部でありステップレポートを使用できる場合は、どのステップが失敗したかを確認し、顧客側に表示されたメッセージを見るために関連するスクリーンショットを参照します。

HAR ファイルをチェックして、失敗が 1 つ以上のリクエストで発生しているかどうかを確認することもできます。ログを使用すると、失敗したリクエストとエラーの詳細について、より深く掘り下げることができます。さらに、これらのアーティファクトと成功した Canary 実行のアーティファクトを比較すれば、問題の特定が可能です。

CloudWatch Synthetics は、デフォルトで、UI Canary の各ステップのスクリーンショットをキャプチャします。ただし、スクリーンショットを無効にするようにスクリプトが構成されている場合もあり得ます。その場合は、デバッグ中にスクリーンショットを再度有効にします。同様に、API Canary のデバッグ中に、HTTP リクエストと応答ヘッダー、あるいは本文を表示する必要性が生じることもあります。このデータをレポートに含める方法については、「executeHttpStep(stepName, requestOptions, [callback], [stepConfig])」を参照してください。

- アプリケーションに最近デプロイしている場合には、一度ロールバックし、後にデバッグを行います。
- エンドポイントに手動で接続し、同じ問題を再現できるかどうかを確認します。

トピック

- Lambda 環境の更新後に Canary が失敗する
- Canary が AWS WAF によってブロックされている
- 要素が表示されるのに時間がかかる
- ノードが表示されない、または page.click() での HTMLElement ではない
- アーティファクトを S3 にアップロードできない (Exception: Unable to fetch S3 bucket location: Access Denied)
- Error: Protocol error (Runtime.callFunctionOn): Target closed。
- Canary Failed。Error: No datapoint - Canary Shows timeout error
- 内部エンドポイントにアクセスしたい
- Canary ランタイムバージョンのアップグレードおよびダウングレードの問題
- クロスオリジンリクエスト共有 (CORS) の問題
- Canary 競合状態に関する問題
- VPC での Canary のトラブルシューティング

Lambda 環境の更新後に Canary が失敗する

CloudWatch Synthetics Canary は、アカウントに Lambda 関数として実装されます。これらの Lambda 関数は、セキュリティ更新プログラム、バグ修正、その他の改善を含む Lambda ランタイムの定期的な更新の対象です。Lambda は、既存の関数との後方互換性を備えたランタイム更新を提供するように努めていますが、ソフトウェアパッチと同様に、ランタイム更新が既存の関数に悪影響を及ぼす状況がまれに発生します。Canary が Lambda ランタイム更新の影響を受けたと思われる場合は、(サポートされているリージョン内で) Lambda ランタイム管理手動モードを使用して、Lambda ランタイムバージョンを一時的にロールバックできます。そうすることで、Canary 関数が引き続き機能し、中断が最小限に抑えられるので、最新のランタイムバージョンに戻る前に非互換性を修正する時間を確保できます。

Lambda ランタイムの更新後に Canary が失敗した場合、最良の解決策は、最新の Synthetics ランタイムのいずれかにアップグレードすることです。最新のランタイムの詳細については、「Synthetics のランタイムバージョン」を参照してください。

別の方法として、Lambda ランタイム管理コントロールが使用可能なリージョンでは、ランタイム管理コントロールの手動モードを使用して、Canary を古い Lambda マネージドランタイムに戻すことができます。手動モードは、AWS CLI または Lambda コンソールを使用して、以下のセクションのステップに従って設定できます。

> **Warning**
>
> ランタイム設定を手動モードに変更すると、Lambda 関数は自動モードに戻るまで自動セキュリティ更新を受信しなくなります。この間、Lambda 関数はセキュリティの脆弱性の影響を受けやすくなる可能性があります。

前提条件

- jq のインストール
- AWS CLI の最新バージョンをインストールします。詳細については「AWS CLI のインストールと更新の手順」を参照してください。

ステップ 1: Lambda 関数 ARN を取得する

以下のコマンドを実行して、レスポンスから `EngineArn` フィールドを取得します。この `EngineArn` は、Canary に関連付けられている Lambda 関数の ARN です。この ARN は次のステップで使用します。

```
aws synthetics get-canary --name my-canary | jq '.Canary.EngineArn'
```

EngingArn の出力例:

```
"arn:aws:lambda:us-west-2:123456789012:function:cwsyn-my-canary-dc5015c2-db17-4cb5-
afb1-EXAMPLE991:8"
```

ステップ 2: 最後の正常な Lambda ランタイムバージョン ARN を取得する

Canary が Lambda ランタイム更新の影響を受けたかどうかを理解するには、ログの Lambda ランタイムバージョンの ARN 変更日時が、Canary への影響が確認された日時にと一致するかどうかを確認します。一致しない場合、問題の原因は Lambda ランタイムの更新ではない可能性があります。

Canary が Lambda ランタイム更新の影響を受けている場合は、以前使用していた、正常に動作する Lambda ランタイムバージョンの ARN を特定する必要があります。「Identifying runtime version changes」の手順に従って、以前のランタイムの ARN を見つけます。ランタイムバージョン ARN を記録してステップ 3 に進み、ランタイム管理設定を設定します。

Canary が Lambda 環境の更新の影響をまだ受けていない場合は、現在使用している Lambda ランタイムバージョンの ARN を確認できます。以下のコマンドを実行して、レスポンスから Lambda 関数の RuntimeVersionArn を取得します。

```
aws lambda get-function-configuration \
--function-name "arn:aws:lambda:us-west-2:123456789012:function:cwsyn-my-canary-
dc5015c2-db17-4cb5-afb1-EXAMPLE991:8" | jq '.RuntimeVersionConfig.RuntimeVersionArn'
```

RuntimeVersionArn の出力例:

```
"arn:aws:lambda:us-
west-2::runtime:EXAMPLE647b82f490a45d7ddd96b557b916a30128d9dcab5f4972911ec0f"
```

ステップ 3: Lambda ランタイム管理設定を更新する

AWS CLI または Lambda コンソールを使用して、ランタイム管理設定を更新できます。

AWS CLI を使用して Lambda ランタイム管理設定の手動モードを設定するには

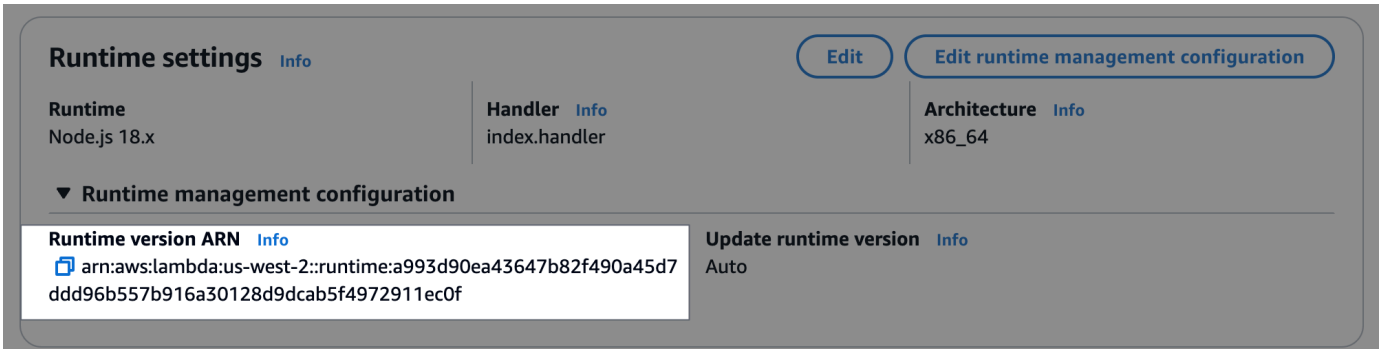
以下のコマンドを入力して、Lambda 関数のランタイム管理を手動モードに変更します。*function-name* と *qualifier* は、ステップ 1 で確認した値の Lambda 関数 ARN と Lambda

関数のバージョン番号にそれぞれ置き換えてください。また、*runtime-version-arn* もステップ 2 で確認したバージョン ARN に置き換えます。

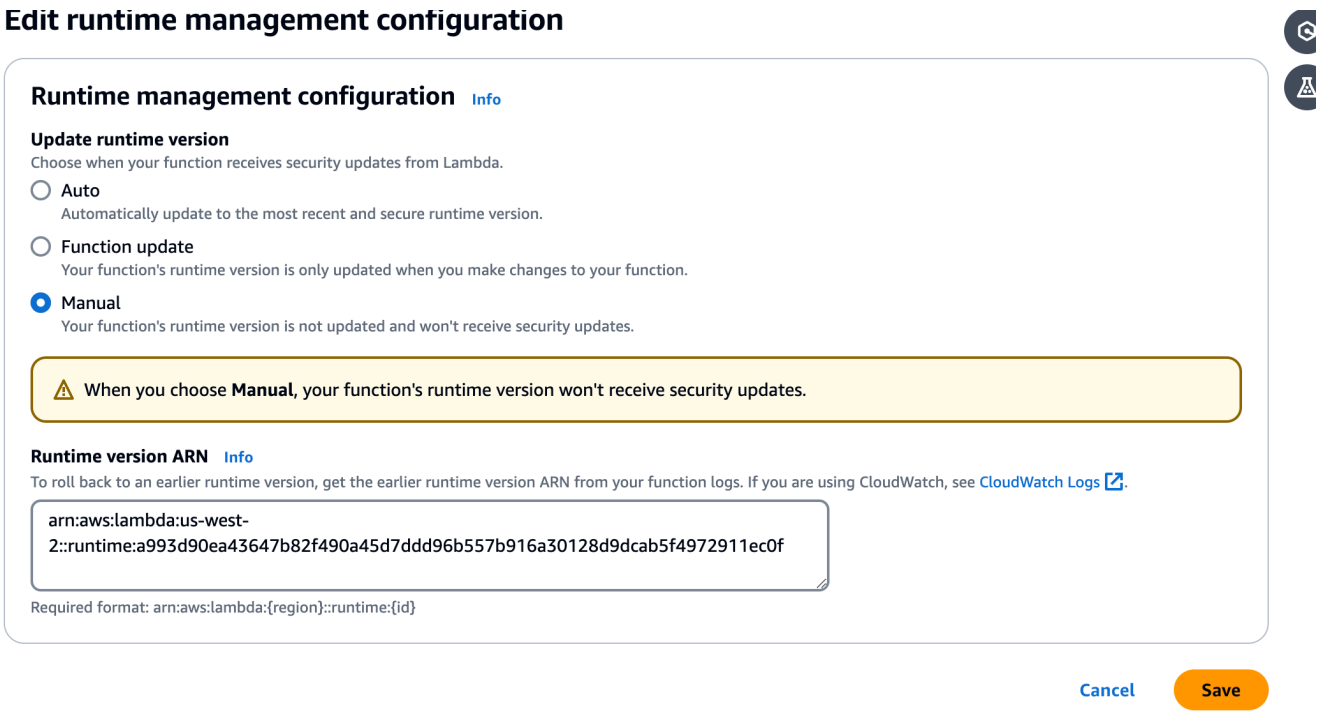
```
aws lambda put-runtime-management-config \
    --function-name "arn:aws:lambda:us-west-2:123456789012:function:cwsyn-my-canary-dc5015c2-db17-4cb5-afb1-EXAMPLE991" \
    --qualifier 8 \
    --update-runtime-on "Manual" \
    --runtime-version-arn "arn:aws:lambda:us-west-2::runtime:a993d90ea43647b82f490a45d7ddd96b557b916a30128d9dcab5f4972911ec0f"
```

Lambda コンソールを使用して Canary を手動モードに変更するには

1. AWS Lambda コンソールを https://console.aws.amazon.com/lambda/ で開きます。
2. [バージョン] タブで ARN に対応するバージョン番号リンクを選択し、[コード] タブを選択します。
3. [ランタイム設定] までスクロールし、[ランタイム管理設定] を展開して、[ランタイムバージョン ARN]をコピーします。



4. [ランタイム管理設定を編集]、[手動] の順に選択し、前にコピーしたランタイムバージョン ARN を[ランタイムバージョン ARN] フィールドに貼り付けます。次に、[Save] (保存) を選択します。



Canary が AWS WAF によってブロックされている

AWS WAF が Canary をブロックしないようにするには、文字列 `CloudWatchSynthetics` を許可する AWS WAF 文字列一致条件を設定します。詳細については、AWS WAF ドキュメントの「Working with string match conditions」を参照してください。

要素が表示されるのに時間がかかる

ログとスクリーンショットを分析した後、要素が画面に表示されるまで待機しているスクリプトがタイムアウトしてしまう場合には、関連するスクリーンショットを参照して、要素がページに表示されているかどうかを確認します。`xpath` に誤りがないことを確認します。

Puppetteer 関連の問題については、Puppeteer の GitHub ページ、またはインターネットのフォーラムを参照してください。

ノードが表示されない、または page.click() での HTMLElement ではない

ノードが非表示になっているか、`page.click()` での `HTMLElement` ではない場合は、要素を参照するために使用している `xpath` を確認します。また、要素が画面の下部に表示されている場合は、ビューポートを調整します。CloudWatch Synthetics では、デフォルトで 1920 × 1080 のビューポートが使用されます。ビューポートの設定変更は、ブラウザの起動時に行うことができます。または Puppeteer の `page.setViewport` 関数を使用しても変更できます。

アーティファクトを S3 にアップロードできない (Exception: Unable to fetch S3 bucket location: Access Denied)

Amazon S3 でのエラーが原因で Canary が失敗した場合は、アクセス許可に問題が生じるために、Canary 用に作成されたスクリーンショットやログ、またはレポートなどを、CloudWatch Synthetics がアップロードできなくなります。以下をチェックしてください:

- `s3:ListAllMyBuckets` アクセス許可、適切な Amazon S3 バケットへの `s3:GetBucketLocation` アクセス許可、および Canary がアーティファクトを保存するバケットへの `s3:PutObject` アクセス許可が、Canary の IAM ロールに付与済みであることを確認します。Canary がビジュアルモニターリングを実行する場合、ロールにはバケットへの `s3:GetObject` アクセス許可も必要です。canary を VPC エンドポイントのある VPC にデプロイしている場合は、Amazon VPC S3 ゲートウェイエンドポイントポリシーにもこれらと同じアクセス権限が必要です。
- Canary が、暗号化用に標準の AWS 管理キー (デフォルト) ではなく AWS KMS 顧客管理キーを使用している場合、その Canary の IAM ロールは、対象のキーを使用して暗号化または復号を行うための許可を持っていないことがあります。詳細については、「Canary アーティファクトの暗号化」を参照してください。
- バケットポリシーで、Canary が使用する暗号化メカニズムが許可されていない可能性があります。例えば、バケットポリシーで特定の暗号化メカニズムまたは KMS キーの使用が必須となっている場合は、Canary に対しても、同じ暗号化モードを選択する必要があります。

Canary がビジュアルモニターリングを実行する際の詳細については、「ビジュアルモニターリング使用時のアーティファクトの場所と暗号化の更新」を参照してください。

Error: Protocol error (Runtime.callFunctionOn): Target closed。

このエラーは、ページまたはブラウザを閉じた後に、ネットワークリクエストが送られた場合に表示されます。非同期オペレーションのための待機を行っていない可能性があります。スクリプトを実行した後、CloudWatch Synthetics はブラウザを閉じます。ブラウザを閉じた後に非同期オペレーションを実行すると、`target closed error` の原因となる可能性があります。

Canary Failed。Error: No datapoint - Canary Shows timeout error

これは、Canary の実行がタイムアウトしたことを意味します。CloudWatch Synthetics が成功率に関する CloudWatch メトリクスを公開したり、HAR ファイル、ログ、スクリーンショットなどのアーティファクトを更新したりする前に、Canary の実行が停止しています。タイムアウト値が小さすぎる場合は増加することができます。

デフォルトでは、Canary のタイムアウト値はその実行間隔と同じに設定されています。タイムアウト値は、手動で Canary の実行間隔以下になるように調整できます。Canary の実行間隔が小さい場合、タイムアウト値を増やすには実行間隔も大きくする必要があります。CloudWatch Synthetics コンソールを使用して Canary を作成または更新する際に、[スケジュール] で実行間隔とタイムアウト値 の両方を調整します。

Lambda コールドスタートと canary インストルメンテーションの起動にかかる時間を許容するために、canary タイムアウト値が 15 秒以上であることを確認してください。

このエラーが発生した場合、Canary のアーティファクトは、CloudWatch Synthetics コンソール上に表示されなくなります。CloudWatch Logs を使用して、Canary のログを表示できます。

CloudWatch Logs を使用して Canary のログを表示するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. 左側のナビゲーションペインで、[ロググループ] をクリックします。
3. フィルターボックスに Canary 名を入力して、ロググループを検索します。Canary のロググループの名前は、/aws/lambda/cwsyn-***canaryName***-randomId です。

内部エンドポイントにアクセスしたい

Canary が内部ネットワーク上のエンドポイントにアクセスできるようにするには、CloudWatch Synthetics で VPC を使用するようにセットアップすることをお勧めします。詳細については、「VPC で Canary を実行する」を参照してください。

Canary ランタイムバージョンのアップグレードおよびダウングレードの問題

最近、ランタイムバージョン `syn-1.0` から新しいバージョンに Canary をアップグレードした場合は、クロスオリジンリクエスト共有 (CORS) の問題が発生する可能性があります。詳細については、「クロスオリジンリクエスト共有 (CORS) の問題」を参照してください。

最近、Canary を古いランタイムバージョンにダウングレードした場合は、使用している CloudWatch Synthetics 関数が、古い方のランタイムバージョンでも利用可能であることを確認します。例えば、`executeHttpStep` 関数はランタイムバージョン `syn-nodejs-2.2` 以降であれば使用が可能です。関数の可用性を確認するには、「Canary スクリプトの作成」を参照してください。

> **Note**
>
> Canary のランタイムバージョンをアップグレードまたはダウングレードする場合は、まず Canary をクローンし、複製された方の Canary でランタイムバージョンを変更することをお勧めします。新しいランタイムバージョンのクローンが機能することを確認したら、元の Canary のランタイムバージョンを更新し、クローンを削除できます。

クロスオリジンリクエスト共有 (CORS) の問題

UI Canary で、一部のネットワークリクエストが `403` または `net::ERR_FAILED` で失敗する場合は、Canary でアクティブなトレースが有効になっているかどうかを確認し、Puppeteer の `page.setExtraHTTPHeaders` 関数を使用してヘッダーを追加します。この場合、ネットワークリクエストの失敗は、クロスオリジンリクエスト共有 (CORS) に関する制限が原因となっている可能性があります。アクティブなトレースを無効にするか、余分な HTTP ヘッダーを削除することで、これが原因なのかどうかを確認できます。

なぜこれが起こるのですか?

アクティブなトレースを使用すると、送信されるすべてのリクエストに余分なヘッダーが追加されることで、呼び出しの追跡が行われます。トレースヘッダーを追加したことでリクエストヘッダーが変更されるか、Puppeteer の `page.setExtraHTTPHeaders`を使用して余分なヘッダーが追加されると、XMLHttpRequest (XHR) リクエストの CORS チェックが実行されます。

アクティブなトレースを無効にしたり、余分なヘッダーを削除したりしたくない場合は、ウェブアプリケーションを更新しクロスオリジンアクセスを許可します。あるいは、スクリプトから Chrome ブラウザーを起動し、その際に `disable-web-security` フラグを使用することでウェブセキュリティを無効にします。

CloudWatch Synthetics の起動関数を使用すると、CloudWatch Synthetics の起動パラメータをオーバーライドし、追加の `disable-web-security` フラグパラメータを渡すことができます。詳細については、「Node.js Canary スクリプト用のライブラリ関数」を参照してください。

> **Note**
>
> ランタイムバージョン `syn-nodejs-2.1` 以降を使用している場合は、CloudWatch Synthetics に渡す起動パラメータをオーバーライドできます。

Canary 競合状態に関する問題

CloudWatch Synthetics を使用するときに満足できるエクスペリエンスを得るには、Canary 用に記述されたコードがべき等性であることを確認します。そうしないと、まれに、Canary が複数の実行で同じリソースとやりとりを行うと、Canary の実行で競合状態が発生することがあります。

VPC での Canary のトラブルシューティング

VPC 内で Canary を作成または更新した後に問題が発生した場合は、以下のいずれかのセクションを参照して問題のトラブルシューティングを行ってください。

新しい Canary がエラー状態になるか、Canary を更新できない

VPC で実行する Canary を作成した場合に、すぐにエラー状態になったり、VPC で実行するように Canary を更新したりできないときは、Canary のロールに適切なアクセス許可がない可能性があります。Canary を VPC で実行する場合には、アクセス許可として `ec2:CreateNetworkInterface`、`ec2:DescribeNetworkInterfaces`、および `ec2:DeleteNetworkInterface` が必要です。これらのすべてのアクセス許可は、`AWSLambdaVPCAccessExecutionRole` 管理ポリシーに含まれています。詳細については、「実行ロールおよびユーザーアクセス許可」を参照してください。

Canary の作成時にこの問題が発生した場合は、この Canary を削除して新しい Canary を作成する必要があります。CloudWatch コンソールを使用して新しい Canary を作成する場合は、[アクセス許可]、[新しいロールの作成] の順に選択します。Canary の実行に必要なすべてのアクセス許可を備えた新しいロールが作成されます。

この問題が Canary の更新時に発生した場合は、Canary を再度更新して、必要なアクセス許可を備えた新しいロールを指定できます。

「テスト結果が返されませんでした」エラー

Canary に「テスト結果が返されませんでした」というエラーが表示される場合は、次のいずれかの問題が原因である可能性があります。

- VPC にインターネットアクセスがない場合は、VPC エンドポイントを使用して CloudWatch および Amazon S3 へのアクセスを Canary に許可する必要があります。これらのエンドポイントアドレスを正しく解決するには、VPC で [DNS 解決] オプションと [DNS ホスト名] オプションを有効にする必要があります。詳細については、「VPC での DNS の使用」および「インターフェイス VPC エンドポイントでの CloudWatch および CloudWatch Synthetics の使用」を参照してください。

- Canary は、VPC 内のプライベートサブネットで実行する必要があります。これを確認するには、VPC コンソールで [サブネット] ページを開きます。Canary の設定時に選択したサブネットを確認します。インターネットゲートウェイ (igw-) へのパスがある場合、これらはプライベートサブネットではありません。

これらの問題のトラブルシューティングを行うには、Canary のログを参照してください。

Canary のログイベントを表示するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで、[Log groups] (ロググループ) を選択します。
3. Canary のロググループの名前を選択します。ロググループ名は `/aws/lambda/cwsyn-`*`canary-name`* で始まります。

Canary スクリプトのサンプルコード

このセクションに含まれているコードサンプルは、CloudWatch Synthetics の Canary スクリプトで使用できる関数を示しています。

Node.js と Puppeteer のサンプル

Cookie の設定

ウェブサイトは、カスタム機能を提供したり、ユーザーを追跡したりするために Cookie を利用します。CloudWatch Synthetics スクリプトで Cookie を設定することで、このカスタム動作を再現して検証することができます。

例えば、再訪問しているユーザーのために、[Register] (登録) リンクではなく、 [Login] (ログイン) リンクがウェブサイトに表示される場合があります。

```
var synthetics = require('Synthetics');
const log = require('SyntheticsLogger');

const pageLoadBlueprint = async function () {

    let url = "http://smile.amazon.com/";

    let page = await synthetics.getPage();
```

```
    // Set cookies.  I found that name, value, and either url or domain are required
 fields.
    const cookies = [{
      'name': 'cookie1',
      'value': 'val1',
      'url': url
    },{
      'name': 'cookie2',
      'value': 'val2',
      'url': url
    },{
      'name': 'cookie3',
      'value': 'val3',
      'url': url
    }];

    await page.setCookie(...cookies);

    // Navigate to the url
    await synthetics.executeStep('pageLoaded_home', async function (timeoutInMillis =
 30000) {

        var response = await page.goto(url, {waitUntil: ['load', 'networkidle0'],
 timeout: timeoutInMillis});

        // Log cookies for this page and this url
        const cookiesSet = await page.cookies(url);
        log.info("Cookies for url: " + url + " are set to: " +
 JSON.stringify(cookiesSet));
    });

};

exports.handler = async () => {
    return await pageLoadBlueprint();
};
```

デバイスのエミュレーション

さまざまなデバイスをエミュレートするスクリプトを記述して、これらのデバイスでのページの外観と動作を予想できます。

次のサンプルは、iPhone 6 デバイスをエミュレートします。エミュレーションの詳細については、Puppeteer ドキュメントの page.emulate(options) を参照してください。

```
var synthetics = require('Synthetics');
const log = require('SyntheticsLogger');
const puppeteer = require('puppeteer-core');

const pageLoadBlueprint = async function () {

    const iPhone = puppeteer.devices['iPhone 6'];

    // INSERT URL here
    const URL = "https://amazon.com";

    let page = await synthetics.getPage();
    await page.emulate(iPhone);

    //You can customize the wait condition here. For instance,
    //using 'networkidle2' may be less restrictive.
    const response = await page.goto(URL, {waitUntil: 'domcontentloaded', timeout:
 30000});
    if (!response) {
        throw "Failed to load page!";
    }

    await page.waitFor(15000);

    await synthetics.takeScreenshot('loaded', 'loaded');

    //If the response status code is not a 2xx success code
    if (response.status() < 200 || response.status() > 299) {
        throw "Failed to load page!";
    }
};

exports.handler = async () => {
    return await pageLoadBlueprint();
};
```

マルチステップ API Canary

このサンプルコードは、正常系テストケースと異常系テストケースについて同じ API をテストするという 2 つの HTTP ステップで API Canary のデモを実行します。ステップ設定が渡され、リクエ

スト/レスポンスヘッダーのレポートが有効になります。さらに、Authorization ヘッダーと X-Amz-Security-Token にはユーザーの認証情報が含まれているため、非表示になります。

このスクリプトを Canary として使用すると、各ステップの詳細と、ステップの合格/不合格、期間、DNS ルックアップ時間や最初のバイト時間などのパフォーマンスメトリクスなど、関連する HTTP リクエストを表示できます。Canary 実行の 2xx、4xx、および 5xx の数を表示できます。

```
var synthetics = require('Synthetics');
const log = require('SyntheticsLogger');

const apiCanaryBlueprint = async function () {

    // Handle validation for positive scenario
    const validatePositiveCase = async function(res) {
        return new Promise((resolve, reject) => {
            if (res.statusCode < 200 || res.statusCode > 299) {
                throw res.statusCode + ' ' + res.statusMessage;
            }

            let responseBody = '';
            res.on('data', (d) => {
                responseBody += d;
            });

            res.on('end', () => {
                // Add validation on 'responseBody' here if required. For ex, your
 status code is 200 but data might be empty
                resolve();
            });
        });
    };

    // Handle validation for negative scenario
    const validateNegativeCase = async function(res) {
        return new Promise((resolve, reject) => {
            if (res.statusCode < 400) {
                throw res.statusCode + ' ' + res.statusMessage;
            }

            resolve();
        });
    };
```

```
    let requestOptionsStep1 = {
        'hostname': 'myproductsEndpoint.com',
        'method': 'GET',
        'path': '/test/product/validProductName',
        'port': 443,
        'protocol': 'https:'
    };

    let headers = {};
    headers['User-Agent'] = [synthetics.getCanaryUserAgentString(), headers['User-
Agent']].join(' ');

    requestOptionsStep1['headers'] = headers;

    // By default headers, post data and response body are not included in the report
 for security reasons.
    // Change the configuration at global level or add as step configuration for
 individual steps
    let stepConfig = {
        includeRequestHeaders: true,
        includeResponseHeaders: true,
        restrictedHeaders: ['X-Amz-Security-Token', 'Authorization'], // Restricted
 header values do not appear in report generated.
        includeRequestBody: true,
        includeResponseBody: true
    };


    await synthetics.executeHttpStep('Verify GET products API with valid name',
 requestOptionsStep1, validatePositiveCase, stepConfig);

    let requestOptionsStep2 = {
        'hostname': 'myproductsEndpoint.com',
        'method': 'GET',
        'path': '/test/canary/InvalidName(',
        'port': 443,
        'protocol': 'https:'
    };

    headers = {};
    headers['User-Agent'] = [synthetics.getCanaryUserAgentString(), headers['User-
Agent']].join(' ');
```

```
    requestOptionsStep2['headers'] = headers;

    // By default headers, post data and response body are not included in the report
 for security reasons.
    // Change the configuration at global level or add as step configuration for
 individual steps
    stepConfig = {
        includeRequestHeaders: true,
        includeResponseHeaders: true,
        restrictedHeaders: ['X-Amz-Security-Token', 'Authorization'], // Restricted
 header values do not appear in report generated.
        includeRequestBody: true,
        includeResponseBody: true
    };

    await synthetics.executeHttpStep('Verify GET products API with invalid name',
 requestOptionsStep2, validateNegativeCase, stepConfig);

};

exports.handler = async () => {
    return await apiCanaryBlueprint();
};
```

Python と Selenium のサンプル

次のサンプル Selenium コードは、ターゲット要素がロードされていない場合にカスタムエラーメッセージを表示して失敗する Canary です。

```
from aws_synthetics.selenium import synthetics_webdriver as webdriver
from aws_synthetics.common import synthetics_logger as logger
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
from selenium.webdriver.common.by import By

def custom_selenium_script():
    # create a browser instance
    browser = webdriver.Chrome()
    browser.get('https://www.example.com/')
    logger.info('navigated to home page')
    # set cookie
    browser.add_cookie({'name': 'foo', 'value': 'bar'})
    browser.get('https://www.example.com/')
```

```
    # save screenshot
    browser.save_screenshot('signed.png')
    # expected status of an element
    button_condition = EC.element_to_be_clickable((By.CSS_SELECTOR, '.submit-button'))
    # add custom error message on failure
    WebDriverWait(browser, 5).until(button_condition, message='Submit button failed to
 load').click()
    logger.info('Submit button loaded successfully')
    # browser will be quit automatically at the end of canary run,
    # quit action is not necessary in the canary script
    browser.quit()

# entry point for the canary
def handler(event, context):
    return custom_selenium_script()
```

Canary と X-Ray のトレース

syn-nodejs-2.0 以降のランタイムを使用する Canary でアクティブな AWS X-Ray トレースを有効にすることもできます。トレースが有効になっている場合、ブラウザ、AWS SDK、または HTTP または HTTPS モジュールを使用するCanary によって行われたすべての呼び出しに対してトレースが送信されます。トレースを有効にした canary は、X-Ray トレースマップに加え、アプリケーションで有効にした Application Signals にも表示されます。

> **Note**
>
> canary でのX-Ray 追跡のアクティブ化は、アジアパシフィック (ジャカルタ) ではまだサポートされていません。

X-Ray トレースマップでは、canary は新規クライアントノードタイプとして表示されます。Canary ノードにカーソルを合わせると、レイテンシー、リクエスト、および障害に関するデータを表示できます。また、Canary ノードを選択して、ページの下部にさらに多くのデータを表示することもできます。ページのこの領域から、[View in Synthetics (Synthetics で表示)] を選択して CloudWatch Synthetics コンソールにジャンプし、Canary の詳細を表示するか、[View Traces (トレースの表示)] を選択してこの Canary の実行からのトレースの詳細を表示できます。

トレースが有効な Canary にも、その詳細ページに [Tracing (トレース)] タブがあり、Canary の実行からのトレースとセグメントの詳細が表示されます。

トレースを有効にすると、Canary 実行時間が 2.5% ~ 7% 増加します。

トレースが有効になっている Canary は、以下のアクセス許可を持つロールを使用する必要があります。Canary を作成するときにコンソールを使用してロールを作成する場合、これらのアクセス許可が与えられます。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Sid230934",
            "Effect": "Allow",
            "Action": [
                "xray:PutTraceSegments"
            ],
            "Resource": "*"
        }
    ]
}
```

Canary によってトレースを生成すると、料金が発生します。X-Ray 料金の詳細については、AWS X-Ray の料金を参照してください。

VPC で Canary を実行する

VPC のエンドポイントとパブリック内部エンドポイントで Canary を実行できます。VPC で Canary を実行するには、VPC で [DNS 解決] オプションと [DNS ホスト名] オプションの両方を有効にする必要があります。詳細については、「VPC での DNS の使用」を参照してください。

VPC エンドポイントで Canary を実行する場合は、Canary から CloudWatch にメトリクスを送信したり、Amazon S3 にアーティファクトを送信したりする方法を指定する必要があります。VPC がインターネットアクセス用にすでに有効になっている場合は、何の操作も必要ありません。Canary は VPC で実行されますが、インターネットにアクセスしてメトリクスとアーティファクトをアップロードできます。

VPC でインターネットアクセスがまだ有効になっていない場合は、次の 2 つのオプションがあります。

- インターネットアクセスを有効にします。詳細については、次の「VPC で Canary へのインターネットアクセスを許可する」セクションを参照してください。
- VPC をプライベートにしておく場合は、プライベート VPC エンドポイントを介してデータを CloudWatch や Amazon S3 に送信するように Canary を設定できます。まだ作成していない場合

は、CloudWatch の VPC エンドポイント (com.amazonaws.*region*.monitoring) と Amazon S3 のゲートウェイエンドポイントを作成する必要があります。詳細については、「インターフェイス VPC エンドポイントでの CloudWatch および CloudWatch Synthetics の使用」および「Amazon S3 における Amazon VPC エンドポイント」を参照してください。

VPC で Canary へのインターネットアクセスを許可する

VPC Canary へのインターネットアクセスを許可する、または Canary に静的 IP アドレスを割り当てるには、次の手順に従います。

VPC 内の Canary にインターネットアクセスを許可するには

1. VPC のパブリックサブネット内に、NAT ゲートウェイをを作成します。手順については、「NAT ゲートウェイの作成」を参照してください。
2. Canary が起動されるプライベートサブネットのルートテーブルに新しいルートを追加します。次を指定します:
   - [Destination] (送信先) に「**`0.0.0.0/0`**」と入力します。
   - [ターゲット] で [NAT ゲートウェイ] を選択し、作成した NAT ゲートウェイの ID を選択します。
   - [ルートの保存] を選択します。

   ルートテーブルへのルートの追加については、「ルートテーブルのルートの追加と削除」を参照してください。

> **Note**
>
> NAT ゲートウェイへのルートのステータスが、[active] (アクティブ) であることを確認します。NAT ゲートウェイが削除され、そのルートが更新されていない場合は、ステータスが「blackhole」になります。詳細については、「NAT ゲートウェイの使用」を参照してください

Canary アーティファクトの暗号化

CloudWatch Synthetics は、スクリーンショット、HAR ファイル、レポートなどの Canary アーティファクトを Amazon S3 バケットに保存します。デフォルトでは、これらのアーティファクトは保存時に AWS マネージドキーを使用して暗号化されます。詳細については、「カスタマーキーと AWS キー」を参照してください。

また、別の暗号化オプションを使用することも選択できます。CloudWatch Synthetics では、以下がサポートされています。

- SSE-S3 – Amazon S3 マネージドキーを使用したサーバー側の暗号化 (SSE)
- SSE-KMS – AWS KMS カスタマー管理キーを使用したサーバー側の暗号化 (SSE)。

AWS マネージドキーを使用するデフォルトの暗号化オプションを選択する場合には、追加のアクセス許可は必要ありません。

SSE-S3 暗号化を使用するには、Canary を作成または更新する際に、暗号化モードとして SSE_S3 を指定します。この暗号化モードを使用するのであれば、追加のアクセス許可は必要ありません。詳細については、「Amazon S3 が管理する暗号化キーによるサーバー側の暗号化 (SSE-S3) を使用したデータの保護」を参照してください。

AWS KMS のカスタマー管理キーを使用するには、Canary を作成または更新する際に、暗号化モードとして SSE-KMS を指定し、同時にキーの Amazon リソースネーム (ARN) も指定します。また、クロスアカウントの KMS キーを使用することもできます。

カスタマー管理のキーを使用するには、次の設定が必要です。

- Canary の IAM ロールには、自分のキーを使用してアーティファクトを暗号化するためのアクセス許可が必要です。ビジュアルモニターリングを使用している場合は、アーティファクトを復号するためのアクセス許可も、同時に付与する必要があります。

```
{
    "Version": "2012-10-17",
    "Statement": {"Effect": "Allow",
        "Action": [
            "kms:GenerateDataKey",
            "kms:Decrypt"
        ],
        "Resource": "Your KMS key ARN"
```

```
    }
}
```

- IAM ロールにアクセス許可を追加する代わりに、キーポリシーに IAM ロールを追加することもできます。複数の Canary に同じロールを使用する場合は、このアプローチも考慮に入れてください。

```
{
    "Sid": "Enable IAM User Permissions",
    "Effect": "Allow",
    "Principal": {
        "AWS": "Your synthetics IAM role ARN"
    },
    "Action": [
        "kms:GenerateDataKey",
        "kms:Decrypt"
    ],
    "Resource": "*"
}
```

- クロスアカウントの KMS キーを使用している場合は、「他のアカウントのユーザーに KMS キーの使用を許可する」を参照してください。

カスタマー管理キーの使用時に暗号化された Canary アーティファクトを表示する

Canary アーティファクトを表示するには、使用しているカスタマー管理キーを更新して、アーティファクトを表示しているユーザーに対する復号化を AWS KMS に許可します。または、アーティファクトを表示しているユーザーまたは IAM ロールに復号のためのアクセス許可を追加します。

デフォルトの AWS KMS ポリシーでは、アカウント内の IAM ポリシーが、KMS キーへのアクセス許可を付与できるようにします。クロスアカウントの KMS キーを使用している場合は、「Why are cross-account users getting Access Denied errors when they try to access Amazon S3 objects encrypted by a custom AWS KMS key?」を参照してください。

KMS キーが原因でアクセスが拒否される問題のトラブルシューティングについては、「キーアクセスのトラブルシューティング」を参照してください。

ビジュアルモニターリング使用時のアーティファクトの場所と暗号化の更新

ビジュアルモニターリングを実行する CloudWatch Synthetics では、ユーザーのスクリーンショットと、(ベースラインとして選択された実行中に) 取得されたベースラインスクリーンショットとを比較

します。アーティファクトの場所または暗号化オプションを更新する場合は、以下のいずれかを実行する必要があります。

- アーティファクト用の以前の Amazon S3 ロケーションと新しい Amazon S3 ロケーションの両方に対する十分なアクセス許可が、IAM ロールに付与済みであることを確認します。同時に、以前の暗号化手段と新しい暗号化手段の両方、ならびに KMS キーに対するアクセス許可があることも確認します。
- 次に新しいベースラインとする Canary 実行を選択して、新しいベースラインを作成します。このオプションを使用する場合、IAM ロールに付与する必要があるのは、新しいアーティファクトの場所と暗号化オプションに対する、十分なアクセス許可のみです。

新しくベースラインとする実行を選択する際は、この 2 番目のオプションをお勧めします。これにより、Canary で今後使用しないアーティファクトの場所や暗号化オプションへの依存関係が必要なくなります。

例えば Canary が、アーティファクトのアップロードにアーティファクトの場所 A と KMS キー K を使用しているとします。この Canary で、アーティファクトの場所を B に、KMS キー を L に更新する際には、アーティファクトの場所 (A と B) と KMS キー (K と L) のそれぞれ両方に対するアクセス許可が、IAM ロールに付与されている必要があります。あるいは、次にベースラインとする新しい実行を選択することで、アーティファクトの場所 B と KMS キー L に対するアクセス許可を、Canary IAM ロールが持っていることを確認できます。

Canary の統計および詳細の表示

Canary の詳細を表示し、Canary の実行に関する統計を表示できます。

Canary の実行結果に関するすべての詳細を表示するには、十分なアクセス許可を備えたアカウントにログオンしている必要があります。詳細については、「CloudWatch Canary に必要なロールとアクセス許可」を参照してください。

Canary の統計情報および詳細を表示するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで、[Application Signals]、[Synthetics Canary] の順に選択します。

   作成した Canary に関する以下の詳細が表示されます。

   - [Status] は、最新の実行に合格した Canary の数を視覚的に示します。

- [Groups] (グループ) は、作成したグループと、失敗した canary または警戒すべき canary があるグループの数を表示します。
- [Slowest performers] (最も遅いパフォーマー) グループと、最もパフォーマンスの低い canary のあるリージョンを表示します。これらは、グループまたはリージョン内のすべての canary (選択した期間全体) の平均期間を合計し、それをグループまたはリージョンの canary の数で割ることによって計算されます。最も遅いグループのメトリクスを選択すると、最も遅いグループとその canary のみが表示されるようにテーブルがフィルタリングされます。テーブルは平均期間でソートされています。
- ページの下部には、すべての Canary を表示するテーブルがあります。1 つの列には、各 canary に対して作成されたアラームが表示されます。Canary アラームの命名基準に準拠したアラームのみが表示されます。この規格は `Synthetics-Alarm-`*`canaryName`*`-`*`index`* です。CloudWatch コンソールの [Synthetics] セクションで作成した Canary アラームでは、この命名規則が自動的に使用されます。CloudWatch コンソールの [Alarms] (アラーム) セクションまたは AWS CloudFormation を使用してCanary を作成し、この命名規則を使用しない場合、アラームは機能しますが、このリストには表示されません。

3. 1 つの canary の詳細を表示するには、[Canaries] テーブルで canary の名前を選択します。

   その Canary に関する以下の詳細が表示されます。

   - [Availability (アベイラビリティー)] タブには、このCanary の最近の実行に関する情報が表示されます。

     [Canary runs] では、いずれかの行を選択すると、その実行に関する詳細を表示できます。

     グラフの下で、[Steps] (ステップ)、[Screenshot] (スクリーンショット)、[Logs] (ログ)、[HAR file] (HAR ファイル) を選択して、これらのタイプの詳細を表示できます。Canary でアクティブなトレースが有効になっている場合は、[Traces (トレース)] を選択して、Canary の実行からトレース情報を表示することもできます。

     Canary の実行のログは S3 バケットと CloudWatch Logs に保存されます。

     スクリーンショットは、顧客がウェブページを表示した方法を示しています。HAR ファイル (HTTP アーカイブファイル) を使用すると、ウェブページに関する詳細なパフォーマンスデータを表示できます。ウェブリクエストのリストを分析し、項目の読み込み時間などのパフォーマンス問題を検出できます。ログファイルには、Canary の実行とウェブページとのやり取りの記録が表示されます。これらを使用して、エラーの詳細を確認できます。

Canary が `syn-nodejs-2.0-beta` ランタイム以降を使用する場合、ステータスコード、リクエストサイズ、または期間によって HAR ファイルをソートすることができます。

[Steps] (ステップ) タブには、canary のステップ、各ステップのステータス、エラーの理由、ステップ実行後の URL、スクリーンショット、およびステップ実行期間のリストが表示されます。HTTP ステップを持つ API Canary については、ランタイム `syn-nodejs-2.2` 以降を使用している場合は、ステップとそれに対応する HTTP リクエストを表示できます。

[HTTP Requests] (HTTP リクエスト) タブを選択して、Canary によって行われた各 HTTP リクエストのログを表示します。リクエスト/レスポンスヘッダー、レスポンス本文、ステータスコード、エラーとパフォーマンスのタイミング (合計時間、TCP 接続時間、TLS ハンドシェイク時間、最初のバイト時間、およびコンテンツ転送時間) を表示できます。HTTP/HTTPS モジュールを内部で使用するすべての HTTP リクエストは、ここで取得されます。

デフォルトで、API Canary においては、セキュリティ上の理由から、リクエストヘッダー、レスポンスヘッダー、リクエスト本文、およびレスポンス本文はレポートに含まれません。それらを含めるように選択した場合、データは S3 バケットにのみ保存されます。このデータをレポートに含める方法については、「executeHttpStep(stepName, requestOptions, [callback], [stepConfig])」を参照してください。

テキスト、HTML、および JSON のレスポンス本文のコンテンツタイプがサポートされています。テキスト/HTML、テキスト/プレーン、アプリケーション/JSON、アプリケーション/x-amz-json-1.0 のようなコンテンツタイプがサポートされています。圧縮されたレスポンスはサポートされていません。

- [Monitoring (モニターリング)] タブには、この Canary で公開された CloudWatch メトリクスのグラフが表示されます。これらのメトリクスの詳細については、「Canary によって発行される CloudWatch メトリクス」を参照してください。

  Canary が公開する CloudWatch グラフィックの下には、Canary の Lambda コードに関連する Lambda メトリクスのグラフがあります。
- [Configuration (設定)] タブには、Canary に関する設定とスケジュール情報が表示されます。
- [Groups] (グループ) タブには、この canary が関連付けられているグループがあれば、そのグループが表示されます。
- [Tags (タグ)] タブには、Canary に関連付けられたタグが表示されます。

Canary によって発行される CloudWatch メトリクス

Canary は、次のメトリクスを CloudWatchSynthetics 名前空間の CloudWatch に公開します。CloudWatch メトリクスの表示方法の詳細については、「利用可能なメトリクスを表示する」を参照してください

メトリクス	説明
SuccessPercent	成功し、失敗が見つからないこの Canary の実行の割合。 有効なディメンション: CanaryName 有効な統計: Average 単位: パーセント
Duration	Canary 実行の時間 (ミリ秒)。 有効なディメンション: CanaryName 有効な統計: Average 単位: ミリ秒
Errors	Canary がフルスクリプトの実行に失敗した回数。 有効なディメンション: CanaryName 有効な統計: Sum
2xx	応答コード 200～299 で、OK 応答を返した Canary によって実行された、ネットワーク要求の数。 このメトリクスは、ランタイムバージョン syn-nodejs-2.0 以降を使用する UI Canary について、およびランタイムバージョン syn-nodejs-2.2 以降を使用する API Canary について、それぞれレポートされます。 有効なディメンション: CanaryName 有効な統計: Sum

メトリクス	説明
	単位: 個
4xx	応答コード 400～499 で、Error 応答を返した Canary によって実行された、ネットワーク要求の数。 このメトリクスは、ランタイムバージョン syn-nodejs-2.0 以降を使用する UI Canary について、およびランタイムバージョン syn-nodejs-2.2 以降を使用する API Canary について、それぞれレポートされます。 有効なディメンション: CanaryName 有効な統計: Sum 単位: 個
5xx	応答コード 500～599 で、Fault 応答を返した Canary によって実行された、ネットワーク要求の数。 このメトリクスは、ランタイムバージョン syn-nodejs-2.0 以降を使用する UI Canary について、およびランタイムバージョン syn-nodejs-2.2 以降を使用する API Canary について、それぞれレポートされます。 有効なディメンション: CanaryName 有効な統計: Sum 単位: 個
Failed	実行に失敗したCanary 実行の数。これらの失敗は Canary 自体に関連しています。 有効なディメンション: CanaryName 有効な統計: Sum 単位: 個

メトリクス	説明
`Failed requests`	ターゲットウェブサイトの Canary によって実行され、応答なしで失敗した HTTP リクエストの数。 有効なディメンション: CanaryName 有効な統計: Sum 単位: 個
`VisualMonitoringSuccessPercent`	Canary 実行中のベースラインのスクリーンショットに正常に一致したビジュアル比較の割合。 有効なディメンション: CanaryName 有効な統計: Average 単位: パーセント
`VisualMonitoringTotalComparisons`	Canary 実行中に発生したビジュアル比較の合計数。 有効なディメンション: CanaryName 単位: 個

> **Note**
>
> Synthetics ライブラリの `executeStep()` または `executeHttpStep()` メソッドのいずれかを使用する Canary は、各ステップのディメンション `CanaryName` および `StepName` を使用して `SuccessPercent` および `Duration` メトリクスも発行します。

Canary を編集または削除する

既存の Canary を編集または削除できます。

Canary を編集する

Canary を編集すると、スケジュールを変更しなくても、Canary を編集したときにスケジュールがリセットされます。例えば、1 時間ごとに実行される Canary があり、その Canary を編集した場合、その Canary は編集が完了した直後に実行され、その後は 1 時間ごとに実行されます。

Canary を編集または更新するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで、[Application Signals]、[Synthetics Canary] の順に選択します。
3. Canary 名の横にあるボタンを選択し、[Actions (アクション)]、[Edit (編集)] の順に選択します。
4. (オプション) この Canary がスクリーンショットのビジュアルモニターリングを実行し、Canary の次回の実行をベースラインとして設定する場合は、[次回の実行を新しいベースラインとして設定] を選択します。
5. (オプション) この Canary がスクリーンショットのビジュアルモニターリングを実行し、ビジュアルモニターリングからスクリーンショットを削除する場合、またはビジュアル比較時にスクリーンショットの一部を無視するように指定する場合は、[ビジュアルモニターリング] で [ベースラインを編集] を選択します。

   スクリーンショットが表示され、次のいずれかを実行できます。

   - スクリーンショットがビジュアルモニターリングに使用されないようにするには、[ビジュアルテストのベースラインからスクリーンショットを削除] を選択します。
   - スクリーンショットの一部を、ビジュアル比較時に無視するように指定するには、クリックおよびドラッグして無視する領域を描画します。比較中に無視するすべてのエリアについてこれを行ったら、[保存] を選択します。
6. Canary にその他の必要な変更を行い、[保存] を選択します。

Canary を削除する

Canary を削除するときに、Canary が使用および作成した他のリソースも削除するかどうかを選択できます。Canary を削除するときは、次のものも削除する必要があります。

- この Canary で使用する Lambda 関数とレイヤー。これらのプレフィックスは `cwsyn-`*`MyCanaryName`* です。
- この Canary 用に作成された CloudWatch アラーム。これらのアラームの名前は `Synthetics-Alarm-`*`MyCanaryName`* で始まります。アラームの削除の詳細については、「CloudWatch アラームを編集または削除する」を参照してください。
- Amazon S3 オブジェクトとバケット (Canary の結果の場所やアーティファクトの場所など)。

- Canary 用に作成した IAM ロール。これらの名前は `role/service-role/CloudWatchSyntheticsRole-`*`MyCanaryName`* です。
- Canary 用に作成された CloudWatch Logs のロググループ。これらのロググループの名前は `/aws/lambda/cwsyn-`*`MyCanaryName`*`-`*`randomId`* です。

Canary を削除する前に Canary の詳細を表示し、これらの情報を書き留めておきます。これにより、Canary の削除後に正しいリソースを削除できます。

Canary を削除するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで、[Application Signals]、[Synthetics Canary] の順に選択します。
3. Canary が現在、`RUNNING` 状態の場合、停止する必要があります。削除できるのは、`STOPPED`、`READY(NOT_STARTED)`、または `ERROR` 状態の Canary のみです。

   Canary を停止するには、Canary 名の横にあるボタンをクリックし、[Actions] (アクション)、[Stop] (停止) の順に選択します。
4. Canary 名の横にあるボタンを選択し、[Actions (アクション)]、[Delete (削除)] の順に選択します。
5. Canary 用に作成され、Canary が使用している他のリソースも削除するかどうかを選択します。これには、Lambda 関数とそのレイヤー、および Canary の IAM ロールと IAM ポリシーが含まれます。

   Canary の IAM ロールと IAM ポリシーを削除するには、適切なアクセス許可が必要です。詳細については、「CloudWatch Synthetics の AWS マネージド (事前定義された) ポリシー」を参照してください。
6. ボックスに **`Delete`** と入力し、[Delete (削除)] を選択します。
7. このセクションで前述したように、Canary により使用され、Canary 用に作成されたその他のリソースを削除します。

複数の Canary のランタイムを開始、停止、削除、または更新する

1 回のアクションで、最大 5 つの Canary のランタイムを停止、開始、削除、または更新できます。Canary のランタイムを更新すると、Canary が使用する言語とフレームワークで使用可能な最新のランタイムに更新されます。

複数の Canary を選択し、且つそのうちの一部の Canary のみが選択したアクションで有効な状態である場合、アクションはそのアクションが有効な Canary でのみ実行されます。例えば、現在実行中の Canary と実行されていない Canary を選択し、 Canary を起動するように選択した場合、まだ実行されていない Canary が起動しますが、すでに実行されている Canary は影響を受けません。

選択した Canary のいずれもアクションに対して有効でない場合、そのアクションはメニューに表示されません。

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで、[Application Signals]、[Synthetics Canary] の順に選択します。
3. 停止、開始、または削除する Canary の横にあるチェックボックスをオンにします。
4. [Actions]] (アクション) を選択し、[Start] (開始)、[Stop] (停止)、[Delete] (削除)、または [Update Runtime] (ランタイムを更新) を選択します。

Amazon EventBridge による Canary イベントのモニターリング

Amazon EventBridge イベントルールは、Canary がステータスを変更したり、実行を完了したときに通知することができます。EventBridge は、AWS リソースの変更を記述したシステムイベントのストリームをほぼリアルタイムに配信します。CloudWatch Synthetics 機能は、これらのイベントをベストエフォートベースで EventBridge に送信します。ベストエフォート配信とは、CloudWatch Synthetics がすべてのイベントを EventBridge に送信しようとしますが、まれにイベントが配信されない場合があります。EventBridge は、受信したすべてのイベントを少なくとも 1 回は処理します。さらに、イベントリスナーは、イベントが発生した順序でイベントを受信しない場合があります。

> **Note**
>
> Amazon EventBridge は、アプリケーションをさまざまなソースからのデータに接続するために使用できるイベントバスサービスです。詳細については、Amazon EventBridge ユーザーガイドの「Amazon EventBridge とは」を参照してください。

CloudWatch Synthetics は、Canary が状態を変更したり、実行を完了したりしたときにイベントを出力します。CloudWatch Synthetics から送信されたすべてのイベントタイプに一致するイベントパターンを含む EventBridge ルール、または特定のイベントタイプのみに一致する EventBridge ルールを作成できます。Canary がルールをトリガーすると、EventBridge はルールに定義されたターゲットアクションを呼び出します。これにより通知を送信し、イベント情報を取得し、Canary の状態の

変化または Canary の実行の完了に対応して是正措置を講じることができます。例えば、次のユースケースのルールを作成できます。

- Canary 実行が失敗した場合の調査
- Canary が ERROR 状態に入った時期を調べる
- Canary のライフサイクルの追跡
- ワークフローの一部としての Canary 実行の成功または失敗のモニターリング

CloudWatch Synthetics からのイベントの例

このセクションでは、CloudWatch Synthetics のサンプルイベントを示します。イベントフォーマットの詳細については、「EventBridge のイベントとイベントパターン」を参照してください。

Canary 状態変更

このイベントタイプでは、`current-state` および `previous-state` の値は次のようになります。

`CREATING | READY | STARTING | RUNNING | UPDATING | STOPPING | STOPPED | ERROR`

```
{
                "version": "0",
                "id": "8a99ca10-1e97-2302-2d64-316c5dedfd61",
                "detail-type": "Synthetics Canary Status Change",
                "source": "aws.synthetics",
                "account": "123456789012",
                "time": "2021-02-09T22:19:43Z",
                "region": "us-east-1",
                "resources": [],
                "detail": {
                                "account-id": "123456789012",
                                "canary-id": "EXAMPLE-dc5a-4f5f-96d1-989b75a94226",
                                "canary-name": "events-bb-1",
                                "current-state": "STOPPED",
                                "previous-state": "UPDATING",
                                "source-location": "NULL",
                                "updated-on": 1612909161.767,
                                "changed-config": {
                                                "executionArn": {
```

```
                                                                "previous-value":
 "arn:aws:lambda:us-east-1:123456789012:function:cwsyn-events-bb-1-af3e3a05-
dc5a-4f5f-96d1-989EXAMPLE:1",
                                                                "current-value":
 "arn:aws:lambda:us-east-1:123456789012:function:cwsyn-events-bb-1-af3e3a05-
dc5a-4f5f-96d1-989EXAMPLE:2"
                                                        },
                                                        "vpcId": {
                                                                "current-value": "NULL"
                                                        },
                                                        "testCodeLayerVersionArn": {
                                                                "previous-
value": "arn:aws:lambda:us-east-1:123456789012:layer:cwsyn-events-bb-1-af3e3a05-
dc5a-4f5f-96d1-989EXAMPLE:1",
                                                                "current-value":
 "arn:aws:lambda:us-east-1:123456789012:layer:cwsyn-events-bb-1-af3e3a05-
dc5a-4f5f-96d1-989EXAMPLE:2"
                                                        }
                                },
                                "message": "Canary status has changed"
                }
}
```

Canary 実行が正常に完了しました

```
{
                "version": "0",
                "id": "989EXAMPLE-f4a5-57a7-1a8f-d9cc768a1375",
                "detail-type": "Synthetics Canary TestRun Successful",
                "source": "aws.synthetics",
                "account": "123456789012",
                "time": "2021-02-09T22:24:01Z",
                "region": "us-east-1",
                "resources": [],
                "detail": {
                                "account-id": "123456789012",
                                "canary-id": "989EXAMPLE-dc5a-4f5f-96d1-989b75a94226",
                                "canary-name": "events-bb-1",
                                "canary-run-id": "c6c39152-8f4a-471c-9810-989EXAMPLE",
                                "artifact-location": "cw-syn-results-123456789012-us-
east-1/canary/us-east-1/events-bb-1-ec3-28ddbe266797/2021/02/09/22/23-41-200",
                                "test-run-status": "PASSED",
                                "state-reason": "null",
```

```
                                "canary-run-timeline": {
                                                "started": 1612909421,
                                                "completed": 1612909441
                                },
                                "message": "Test run result is generated successfully"
                }
}
```

Canary 実行が正常に完了しませんでした

```
{
                "version": "0",
                "id": "2644b18f-3e67-5ebf-cdfd-bf9f91392f41",
                "detail-type": "Synthetics Canary TestRun Failure",
                "source": "aws.synthetics",
                "account": "123456789012",
                "time": "2021-02-09T22:24:27Z",
                "region": "us-east-1",
                "resources": [],
                "detail": {
                                "account-id": "123456789012",
                                "canary-id": "af3e3a05-dc5a-4f5f-96d1-9989EXAMPLE",
                                "canary-name": "events-bb-1",
                                "canary-run-id": "0df3823e-7e33-4da1-8194-
b04e4d4a2bf6",
                                "artifact-location": "cw-syn-results-123456789012-us-
east-1/canary/us-east-1/events-bb-1-ec3-989EXAMPLE/2021/02/09/22/24-21-275",
                                "test-run-status": "FAILED",
                                "state-reason": "\"Error: net::ERR_NAME_NOT_RESOLVED
 \""
                                "canary-run-timeline": {
                                                "started": 1612909461,
                                                "completed": 1612909467
                                },
                                "message": "Test run result is generated successfully"
                }
}
```

イベントが重複したり、順序が順不同である可能性があります。イベントの順序を決定するには、`time` プロパティを使用します。

EventBridge ルールを作成するための前提条件

CloudWatch Synthetics 用の EventBridge ルールを作成する前に、以下を行う必要があります。

- EventBridge のイベント、ルール、ターゲットに精通しておいてください。
- EventBridge ルールによって呼び出されるターゲットを作成して設定します。ルールは、以下のようなさまざまなタイプのターゲットを呼び出すことができます。
  - Amazon SNS トピック
  - AWS Lambda 関数
  - Kinesis Streams
  - Amazon SQS キュー

詳細については、Amazon EventBridge ユーザーガイドの「Amazon EventBridge とは」および「Amazon EventBridge の開始方法」を参照してください。

EventBridge ルールを作成する (CLI)

以下の例の手順は、`us-east-1` の `my-canary-name` という名前の Canary が実行を完了するか、状態を変更したときに Amazon SNS トピックを公開する EventBridge ルールを作成します。

1. ルールを作成します。

```
aws events put-rule \
  --name TestRule \
  --region us-east-1 \
  --event-pattern "{\"source\": [\"aws.synthetics\"], \"detail\": {\"canary-name\":
 [\"my-canary-name\"]}}"
```

   パターンから省略したプロパティはすべて無視されます。

2. トピックをルールターゲットとして追加します。

   - *topic-arn* を Amazon SNS トピックの Amazon リソースネーム (ARN) に置き換えます。

```
aws events put-targets \
  --rule TestRule \
  --targets "Id"="1","Arn"="topic-arn"
```

> **Note**
>
> Amazon EventBridge にターゲットトピックの呼び出しを許可するには、トピックにリソースベースのポリシーを追加する必要があります。詳細については、Amazon EventBridge ユーザーガイドの「Amazon SNS のアクセス許可」を参照してください。

詳細については、Amazon EventBridge ユーザーガイドの「EventBridge のイベントとイベントパターン」を参照してください。

CloudWatch Evidently での起動と A/B 実験を実行する

Amazon CloudWatch Evidently を使用すると、機能のロールアウト中に、新しい機能を指定した割合のユーザーに提供することによって安全に検証できます。新機能のパフォーマンスをモニターリングすると、ユーザーへのトラフィックを増やしていくタイミングを決定するのに役立ちます。こうすれば、機能を完全に起動する前に、リスクを軽減し、意図しない結果を明確化できます。

また、A/B 実験を実行して、証拠とデータに基づいて機能の設計を決定することもできます。実験では、一度に最大 5 つのバリエーションをテストできます。Evidently では、実験データを収集し、統計的手法を用いて分析します。また、どのバリエーションが優れているかについて、明確なレコメンデーションを提供します。ユーザー向け機能とバックエンド機能の両方をテストできます。

Evidently の料金

Evidently イベントと Evidently 分析ユニットに基づいて、アカウントに課金されます。Evidently イベントには、クリックやページビューなどのデータイベントと、ユーザーに提供する機能のバリエーションを決定する割り当てイベントが含まれます。

Evidently 分析ユニットは、Evidently で作成したルールに基づいて、Evidently イベントから生成されます。分析ユニットは、イベントのルールとの一致の数です。例えば、ユーザーのクリックイベントで、1 つの Evidently 分析ユニット (クリック数) が生成される場合があります。もう 1 つの例は、ユーザーのチェックアウトイベントで、チェックアウト値とカート内のアイテム数という 2 つの Evidently 分析ユニットが生成される場合があります。料金の詳細については、「Amazon CloudWatch の料金」を参照してください。

CloudWatch Evidently は現在、以下のリージョンで利用できます。

- 米国東部 (オハイオ)

- 米国東部 (バージニア北部)
- 米国西部 (オレゴン)
- アジアパシフィック (シンガポール)
- アジアパシフィック (シドニー)
- アジアパシフィック (東京)
- 欧州 (フランクフルト)
- 欧州 (アイルランド)
- 欧州 (ストックホルム)

トピック

Evidently を使用するための IAM ポリシー

CloudWatch Evidently を完全に管理するには、以下のアクセス許可を持つ IAM ユーザーまたはロールとしてサインインする必要があります。

- AmazonCloudWatchEvidentlyFullAccess ポリシー
- ResourceGroupsandTagEditorReadOnlyAccess ポリシー

さらに、Amazon S3 または CloudWatch Logs に評価イベントを保存するプロジェクトを作成するには、次のアクセス許可が必要です。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetBucketPolicy",
                "s3:PutBucketPolicy",
                "s3:GetObject",
                "s3:ListBucket"
            ],
            "Resource": "arn:aws:s3:::*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogDelivery",
                "logs:DeleteLogDelivery",
                "logs:DescribeResourcePolicies",
                "logs:PutResourcePolicy"
            ],
            "Resource": [
                "*"
            ]
        }
    ]
}
```

CloudWatch RUM 統合のための追加のアクセス許可

さらに、Amazon CloudWatch RUM と統合し、CloudWatch RUM メトリクスをモニターリングに使用する Evidently 起動または実験を管理する場合は、AmazonCloudWatchRUMFullAccess ポリシーが必要です。IAM ロールを作成して CloudWatch RUM ウェブクライアントに CloudWatch RUM にデータを送信するアクセス許可を付与するには、次のアクセス許可が必要です。

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

```
            "Effect": "Allow",
            "Action": [
                "iam:CreateRole",
                "iam:CreatePolicy",
                "iam:AttachRolePolicy"
            ],
            "Resource": [
                "arn:aws:iam::*:role/service-role/CloudWatchRUMEvidentlyRole-*",
                "arn:aws:iam::*:policy/service-role/CloudWatchRUMEvidentlyPolicy-*"
            ]
        }
    ]
}
```

Evidently への読み取り専用アクセス許可

Evidently データを表示する必要があるが、Evidently リソースを作成する必要がない他のユーザーには、AmazonCloudWatchEvidentlyReadOnlyAccess ポリシーを付与できます。

プロジェクト、機能、起動、実験を作成する

CloudWatch Evidently を開始するには、機能の起動または A/B 実験に対して、まず プロジェクト を作成します。プロジェクトは、リソースを論理的にグループ化したものです。プロジェクト内では、テストまたは起動をするバリエーションがある機能を作成します。機能は、起動または実験を作成する前、または同時に作成できます。

トピック

- の新規プロジェクトの作成
- クライアント側の評価を使用する - AWS AppConfig を使用
- プロジェクトに機能を追加する
- セグメントを使用してオーディエンスを絞り込む
- 起動の作成
- 実験の作成

の新規プロジェクトの作成

次の手順を使用して、新しい CloudWatch Evidently プロジェクトをセットアップします。

新しい CloudWatch Evidently プロジェクトを作成するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで、[Application Signals]、[Evidently] の順に選択します。
3. [プロジェクトを作成] を選択します。
4. [Project name] (プロジェクト名) には、CloudWatch Evidently コンソール内でこのプロジェクトの識別に使用する名前を入力します。

   オプションとしてプロジェクトの説明を追加することもできます。
5. [Evaluation event storage] (評価イベントストレージ) で、Evidently で収集した評価イベントを保存するかどうかを選択します。これらのイベントを保存しなくても、Evidently はイベントを集約して、Evidently ダッシュボードで表示できるメトリクスやその他の実験データを作成します。詳細については、「プロジェクトデータストレージ」を参照してください。
6. [Use client-side evaluation] (クライアント側の評価を使用する) で、このプロジェクトでクライアント側の評価を有効にするかどうかを選択します。クライアント側の評価では、アプリケーションが EvaluateFeature オペレーションを呼び出す代わりに、ローカルでバリエーションをユーザーセッションに割り当てることができます。これにより、API コールに伴うレイテンシーと可用性のリスクが軽減されます。詳細については、「クライアント側の評価を使用する - AWS AppConfig を使用」を参照してください。

   クライアント側の評価を伴うプロジェクトを作成するには、アクセス許可 `evidently:ExportProjectAsConfiguration` が必要です。

   クライアント側の評価を有効にする場合は、次の操作も行ってください。

   a. 既存の AWS AppConfig アプリケーションを使用するか、新しいアプリケーションを作成するかを選択します。

   b. 既存の AWS AppConfig 環境を使用するか、新しい環境を作成するかを選択します。

      AWS AppConfig のアプリケーションと環境の詳細については、「AWS AppConfig の仕組み」を参照してください。
7. (オプション) このプロジェクトにタグを追加するには、[Tags] (タグ)、[Add new tag] (新しいタグを追加) を選択します。

   [Key] (キー) に、タグの名前を入力します。[値] では、任意でタグに値を追加できます。

   (オプション) 別のタグを追加するには、[Add new tag] (新しいタグを追加) を再度選択します。

詳細については、「AWS リソースのタグ付け」を参照してください。

8. [プロジェクトを作成] を選択します。

クライアント側の評価を使用する - AWS AppConfig を使用

プロジェクトでクライアント側の評価 - AWS AppConfig を使用 (クライアント側の評価) を使用すると、アプリケーションが EvaluateFeature オペレーションを呼び出してバリエーションを割り当てるのではなく、ローカルでバリエーションをユーザーセッションに割り当てることができます。これにより、API コールに伴うレイテンシーと可用性のリスクが軽減されます。

クライアント側の評価を使用するには、AWS AppConfig Lambda の拡張機能を Lambda 関数のレイヤーとしてアタッチし、環境変数を設定します。クライアント側の評価は、ローカルホストのサイドプロセスとして実行されます。次に、`localhost` に対して [EvaluationFeature] および [PutProjectEvent] オペレーションを呼び出すことができます。クライアント側の評価プロセスでは、バリエーションの割り当て、キャッシュ、およびデータ同期の処理が行われます。AWS AppConfig の詳細については、「AWS AppConfig の仕組み」を参照してください。

AWS AppConfig と統合する場合、Evidently に対して AWS AppConfig アプリケーション ID と AWS AppConfig 環境 ID を指定します。Evidenly のプロジェクト間では、同じアプリケーション ID と環境 ID を使用できます。

クライアント側の評価を有効にしたプロジェクトを作成すると、Evidently はそのプロジェクトの AWS AppConfig 設定プロファイルを作成します。各プロジェクトの設定プロファイルは異なります。

クライアント側の評価のアクセスコントロール

Evidently のクライアント側の評価では、他の Evidently とは異なるアクセスコントロールメカニズムを使用しています。このことを理解していただき、適切なセキュリティ対策を実施することを強くお勧めします。

Evidently では、ユーザーが個々のリソースで実行できるアクションを制限する IAM ポリシーを作成できます。例えば、[EvaluateFeature] アクションを持つことを禁止するユーザーロールを作成することができます。IAM ポリシーで制御できる Evidently アクションの詳細については、「Actions defined by Amazon CloudWatch Evidently」(Amazon CloudWatch Evidentlyで定義されるアクション) を参照してください。

クライアント側の評価モデルでは、プロジェクトのメタデータを使用する Evidently 機能をローカルで評価できます。クライアント側の評価が有効になっているプロジェクトのユーザーは、ロー

カルホストのエンドポイントに対して [EvaluateFeature] API を呼び出すことができますが、この API コールは Evidently に到達せず、Evidenly サービスの IAMポリシーによって認証されません。この呼び出しは、ユーザーが [EvaluateFeature] アクションを使用する IAM アクセス許可を持っていない場合でも成功します。ただし、エージェントが評価イベントやカスタムイベントをバッファリングし、Evidently に非同期的にデータをオフロードするためには、ユーザーにはアクセス許可 [PutProjectEvents] が必要です。

また、ユーザーは、クライアント側の評価を使用するプロジェクトを作成するためのアクセス許可 `evidently:ExportProjectAsConfiguration` を持っている必要があります。これにより、クライアント側の評価中に呼び出される [EvaluateFeature] アクションへのアクセスを制御しやすくなります。

クライアント側の評価のセキュリティモデルが、他の Evidently に設定したポリシーを覆す可能性があるため、アクセスコントロールは慎重に行ってください。アクセス許可 `evidently:ExportProjectAsConfiguration` を持つユーザーは、IAM ポリシーで [EvaluateFeature] アクションが明示的に拒否されている場合でも、クライアント側の評価を有効にしたプロジェクトを作成し、[EvaluateFeature] アクションを使用してそのプロジェクトでクライアント側の評価を行うことができます。

Lambda の使用を開始する

Evidently では現在、AWS Lambda 環境を使用するクライアント側の評価がサポートされています。開始するには、まず、どの AWS AppConfig アプリケーションと環境を使用するかを決めます。既存のアプリケーションと環境を選択するか、または新しいアプリケーションと環境を作成するかを選択します。

次のサンプル AWS AppConfig AWS CLI コマンドはアプリケーションと環境を作成します。

```
aws appconfig create-application --name YOUR_APP_NAME
```

```
aws appconfig create-environment --application-id YOUR_APP_ID --
name YOUR_ENVIRONMENT_NAME
```

次に、これらの AWS AppConfig リソースを使用して Evidently プロジェクトを作成します。詳細については、「 の新規プロジェクトの作成」を参照してください。

クライアント側の評価は、Lambda レイヤーを使用することで Lambda でサポートされます。これは `AWS-AppConfig-Extension` の一部であるパブリックレイヤーで、AWS AppConfig サービスに

よって作成されたパブリック AWS AppConfig 拡張機能です。Lambda レイヤーの詳細については、「レイヤー」を参照してください。

クライアント側の評価を使用するには、このレイヤーを Lambda 関数に追加し、アクセス許可と環境変数を設定する必要があります。

Evidently クライアント側の評価 Lambda レイヤーを Lambda 関数に追加して設定するには

1. Lambda 関数を作成します (まだ作成していない場合)。
2. クライアント側の評価レイヤーを関数に追加します。その ARN を指定するか、まだ選択していない場合は AWS レイヤーのリストから選択することができます。詳細については、「関数を設定してレイヤーを使用する」と「AWS AppConfig Lambda 拡張機能の使用可能なバージョン」を参照してください。
3. [EvidentlyAppConfigCachingAgentPolicy] という名前の IAM ポリシーを以下の内容で作成し、関数の実行ロールにアタッチします。詳細については、「Lambda 実行ロール」を参照してください。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": [
                "appconfig:GetLatestConfiguration",
                "appconfig:StartConfigurationSession",
                "evidently:PutProjectEvents"
            ],
            "Resource": "*"
        }
    ]
}
```

4. Lambda 関数に必要な環境変数 `AWS_APPCONFIG_EXTENSION_EVIDENTLY_CONFIGURATIONS` を追加します。この環境変数は、Evidently プロジェクトと AWS AppConfig リソース間のマッピングを指定します。

   1 つの Evidently プロジェクトでこの関数を使用している場合は、環境変数の値を次のように設定します: `applications/`*`APP_ID`*`/environments/`*`ENVIRONMENT_ID`*`/configurations/`*`PROJECT_NAME`*

この関数を複数の Evidently プロジェクトで使用する場合は、次の例のようにコンマを使用して値を区切ります: `applications/APP_ID_1/environments/ENVIRONMENT_ID_1/configurations/PROJECT_NAME_1, applications/APP_ID_2/environments/ENVIRONMENT_ID_2/configurations/PROJECT_NAME_2`

5. (オプション) 他の環境変数を設定します。詳細については、「設定: AWS AppConfig AppConfigLambda の拡張」を参照してください。
6. アプリケーションで、`EvaluateFeature` を `localhost` に送信して Evidently の評価をローカルに入手します。

   Python の例

```
import boto3
from botocore.config import Config

def lambda_handler(event, context):
    local_client = boto3.client(
        'evidently',
        endpoint_url="http://localhost:2772",
        config=Config(inject_host_prefix=False)
    )
    response = local_client.evaluate_feature(
        project=event['project'],
        feature=event['feature'],
        entityId=event['entityId']
    )
    print(response)
```

   Node.js の例:

```
const AWS = require('aws-sdk');
const evidently = new AWS.Evidently({
    region: "us-west-2",
    endpoint: "http://localhost:2772",
    hostPrefixEnabled: false
});

exports.handler = async (event) => {

    const evaluation = await evidently.evaluateFeature({
        project: 'John_ETCProject_Aug2022',
```

```
        feature: 'Feature_IceCreamFlavors',
        entityId: 'John'
    }).promise()

    console.log(evaluation)
    const response = {
        statusCode: 200,
        body: evaluation,
    };
    return response;
};
```

Kotlin の例:

```
String localhostEndpoint = "http://localhost:2772/"
public AmazonCloudWatchEvidentlyClient getEvidentlyLocalClient() {
    return AmazonCloudWatchEvidentlyClientBuilder.standard()

 .withEndpointConfiguration(AwsClientBuilder.EndpointConfiguration(localhostEndpoint,
 region))

 .withClientConfiguration(ClientConfiguration().withDisableHostPrefixInjection(true))
        .withCredentials(credentialsProvider)
        .build();
}

AmazonCloudWatchEvidentlyClient evidently = getEvidentlyLocalClient();

// EvaluateFeature via local client.
EvaluateFeatureRequest evaluateFeatureRequest = new
 EvaluateFeatureRequest().builder()
   .withProject(${YOUR_PROJECT}) //Required.
   .withFeature(${YOUR_FEATURE}) //Required.
   .withEntityId(${YOUR_ENTITY_ID}) //Required.
   .withEvaluationContext(${YOUR_EVAL_CONTEXT}) //Optional: a JSON object of
 attributes that you can optionally pass in as part of the evaluation event sent to
 Evidently.
   .build();

EvaluateFeatureResponse evaluateFeatureResponse =
 evidently.evaluateFeature(evaluateFeatureRequest);

// PutProjectEvents via local client.
```

```
PutProjectEventsRequest putProjectEventsRequest = new
 PutProjectEventsRequest().builder()
    .withData(${YOUR_DATA})
    .withTimeStamp(${YOUR_TIMESTAMP})
    .withType(${YOUR_TYPE})
    .build();

PutProjectEvents putProjectEventsResponse =
 evidently.putProjectEvents(putProjectEventsRequest);
```

クライアントが Evidently にデータを送信する頻度を設定する

クライアント側の評価が Evidently にデータを送信する頻度を指定するには、オプションで 2 つの環境変数を設定できます。

- `AWS_APPCONFIG_EXTENSION_EVIDENTLY_EVENT_BATCH_SIZE` は Evidently に送信する前にバッチ処理するプロジェクトごとのイベント数を指定します。有効な範囲は 1～50 の整数で、デフォルトは 40 です。
- `AWS_APPCONFIG_EXTENSION_EVIDENTLY_BATCH_COLLECTION_DURATION` は Evidently に送信する前にイベントを待機する時間を秒単位で指定します。デフォルトは 30 です。

トラブルシューティング

以下の情報を参考にして、CloudWatch Evidently のクライアント側の評価 (AWS AppConfig を使用) を実行する際のトラブルシューティングを行います。

EvaluateFeature オペレーションの呼び出し中にエラー (BadRequestException) が発生しました: 提供されたパスでは HTTP メソッドがサポートされていません。

環境変数が正しく設定されていない可能性があります。例えば、環境変数名として、`AWS_APPCONFIG_EXTENSION_EVIDENTLY_CONFIGURATIONS` ではなく `EVIDENTLY_CONFIGURATIONS` を使用した可能性があります。

ResourceNotFoundException: デプロイが見つかりません

プロジェクトメタデータの更新が AWS AppConfig にデプロイされていません。クライアント側の評価に使用した AWS AppConfig 環境で、アクティブなデプロイがあるかどうかを確認します。

ValidationException: プロジェクトで Evidently が設定されていません

`AWS_APPCONFIG_EXTENSION_EVIDENTLY_CONFIGURATIONS` 環境変数に間違ったプロジェクト名が設定されている可能性があります。

プロジェクトに機能を追加する

CloudWatch Evidently の機能 は、起動する機能、またはバリエーションをテストする機能を表しています。

機能を追加する前に、プロジェクトを作成する必要があります。詳細については、「 の新規プロジェクトの作成」を参照してください。

プロジェクトに機能を追加するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで、[Application Signals]、[Evidently] の順に選択します。
3. プロジェクトの名前を選択します。
4. [Add feature] (機能を追加) を選択します。
5. [Feature name] (機能名) に、このプロジェクト内でこの機能を識別するために使用する名前を入力します。

   オプションとして機能の説明を追加することもできます。
6. [Feature variations] (機能のバリエーション) では、[Variation type] (バリエーションの型) として [Boolean]、[Long]、[Double]、または [String] を選択します。詳細については、「バリエーションの型」を参照してください。
7. 機能に最大 5 つのバリエーションを追加します。各バリエーションの [Value] (値) は、選択した Variation type (バリエーションの型) として有効である必要があります。

   デフォルトにするバリエーションの 1 つを指定します。これは、他のバリエーションが比較対象とするベースラインであり、現在ユーザーに提供されているバリエーションであることが望ましいです。これは、この機能の起動や実験に加わっていないユーザーに提供されるバリエーションでもあります。
8. [Sample code] (サンプルコード) を選択します。コード例は、バリエーションをセットアップしてそこにユーザーセッションを割り当てるためにアプリケーションに追加する必要があるものを示しています。コードには JavaScript、Java、および Python を選択できます。

今すぐアプリケーションにコードを追加する必要はありませんが、起動または実験を開始する前に、追加する必要があります。

詳細については、「アプリケーションにコードを追加する」を参照してください。

9. (オプション) 特定のユーザーに常に特定のバリエーションが表示されるように指定するには、[Overrides] (オーバーライド)、[Add override] (オーバーライドを追加) を選択します。次に、ユーザー ID、アカウント ID、またはその他の識別子を [Identifier] (識別子) に入力してユーザーを指定し、そのユーザーに表示されるバリエーションを指定します。

   これは、自分のテストチームのメンバーや他の内部ユーザーに特定のバリエーションが必ず表示されるようにする場合に便利です。オーバーライドが割り当てられているユーザーのセッションは、起動または実験メトリクスに寄与しません。

   [オーバーライドを追加] を再度選択すると、最大 20 人のユーザーに対してこれを繰り返すことができます。

10. (オプション) この機能にタグを追加するには、[Tags] (タグ)、[Add new tag] (新しいタグを追加) を選択します。

    [Key] (キー) に、タグの名前を入力します。[値] では、任意でタグに値を追加できます。

    (オプション) 別のタグを追加するには、[Add new tag] (新しいタグを追加) を再度選択します。

    詳細については、「AWS リソースのタグ付け」を参照してください。

11. [Add feature] (機能を追加) を選択します。

バリエーションの型

機能を作成してバリエーションを定義するときは、バリエーションの型を選択する必要があります。可能な型は次のとおりです。

- ブール値
- 長整数
- 倍精度浮動小数点数
- 文字列

バリエーションの型で、コード内の異なるバリエーションを区別する方法を設定します。バリエーションの型を使用すると、CloudWatch Evidently の実装を簡素化し、起動や実験で機能を変更するプロセスを簡素化することもできます。

例えば、バリエーションを長整数型にして機能を定義する場合、バリエーションを区別するために指定する整数は、コードに直接渡される数値になります。ボタンのピクセルサイズをテストするとします。バリエーションの型の値は、各バリエーションで使用されるピクセル数になります。各バリエーションのコードは、バリエーションの型の値を読み取り、それをボタンのサイズとして使用できます。他のコードを変更せずに、バリエーションの値に使用される数値を変更すれば、新しいボタンサイズをテストできます。

機能内でバリエーションの型の値を設定する場合、CloudWatch Evidently の最初の試行として A/A テストを行う場合や、その他の理由がある場合を除き、同じ値を複数のバリエーションに割り当てないようにしてください。

Evidently は、JSON を型としてネイティブサポートしていませんが、文字列のバリエーション型で JSON を渡し、コード内でその JSON を解析できます。

セグメントを使用してオーディエンスを絞り込む

オーディエンスセグメントを定義して起動や実験に使用できます。セグメントとは、1 つまたは複数の特徴を共有する、オーディエンスの一部分です。例えば、Chrome ブラウザのユーザー、ヨーロッパのユーザー、ヨーロッパの Firefox ブラウザのユーザーで、年齢などアプリケーションが収集する他の条件にも適合するユーザーなどです。

実験でセグメントを使用すると、セグメントの条件に一致するユーザーのみを評価するように実験が制限されます。起動で 1 つまたは複数のセグメントを使用すると、異なるオーディエンスセグメントに対して異なるトラフィック分割を定義できます。

セグメントルールのパターンの構文

セグメントを作成するには、セグメントルールのパターンを定義します。ユーザーセッションがそのセグメントに含まれるかどうかを評価するために使用する属性を指定します。作成したパターンは、Evidently がユーザーセッション内で見つけた `evaluationContext` の値と比較されます。詳細については、「EvaluateFeature の使用」を参照してください。

セグメントルールのパターンを作成するには、パターンに一致させたいフィールドを指定します。また、`And`、`Or`、`Not`、`Exists` などのロジックもパターンで使用できます。

evaluationContext がパターンと一致するためには、evaluationContext がルールパターンのすべての部分と一致する必要があります。Evidently では、ルールパターンに含まれていない evaluationContext のフィールドは無視されます。

ルールパターンが一致する値は、JSON のルールに従います。二重引用符 (") で囲んだ文字列、数値、およびキーワード true、false、および null を含めることができます。

文字列の場合、Evidently では文字単位の厳密な一致が使用され、大文字の小文字化など文字列の正規化は行われません。そのため、ルールの一致では大文字と小文字が区別されます。例えば、evaluationContext に browser 属性が含まれていても、ルールパターンが Browser をチェックするため、一致しません。

数値の場合、Evidently では文字列表現を使用します。たとえば、300、300.0、3.0e2 は等しいとはみなされません。

evaluationContext と一致させるルールパターンを記述するときは、TestSegmentPattern API または test-segment-pattern CLI コマンドを使用して、パターンが正しい JSON と一致することをテストできます。詳細については、TestSegmentPattern を参照してください。

以下は、Evidently セグメントパターンで使用可能なすべての比較演算子をまとめたものです。

比較	例	ルール構文
Null	UserId が Null	`{ "UserID": [ null ] }`
空	LastName が空白	`{ "LastName": [""] }`
等しい	ブラウザが「Chrome」	`{ "Browser": [ "Chrome" ] }`
And	国が「フランス」および端末が「モバイル」	`{`

比較	例	ルール構文
		"Country": ["France"], "Device": ["Mobile"] }
または (1 つの属性の複数の値)	ブラウザが「クローム」または「Firefox」	{ "Browser": ["Chrome", "Firefox"] }
または (異なる属性)	ブラウザが「Safari」またはデバイスが「タブレット」	{ "$or": [{"Browser": ["Safari"]}, {"Device": ["Tablet" }]] }
Not	ブラウザが「Safari」以外	{ "Browser": [{ "anything-but": ["Safari"] }] }
数値 (等しい)	Price が 100	{ "Price": [{ "numeric": ["=", 100] }] }
数値 (範囲)	Price が 10 より大きく 20 以下	{ "Price": [{ "numeric": [">", 10, "<=", 20] }] }

比較	例	ルール構文
存在する	Age フィールドが存在する	`{ "Age": [ { "exists": true } ] }`
存在しない	Age フィールドが存在しない	`{ "Age": [ { "exists": false } ] }`
プレフィックスで始まる	リージョンが米国にある	`{ "Region": [ {"prefix": "us-" } ] }`
サフィックスで終わる	場所に「West」のサフィックスが付く	`{ "Region": [ {"suffix": "West" } ] }`

セグメントルールの例

次のすべての例では、ルールパターンで使用しているものと同じフィールドラベルと値で `evaluationContext` の値を渡しているものとします。

次の例は、`Browser` が Chrome または Firefox で、`Location` が米国西部の場合に一致します。

```
{
  "Browser": ["Chrome", "Firefox"],
  "Location": ["US-West"]
}
```

次の例は、`Browser` が Chrome 以外のブラウザで、`Location` が US で始まり、Age フィールドが存在する場合に一致します。

```
{
  "Browser": [ {"anything-but": ["Chrome"]}],
  "Location": [{"prefix": "US"}],
  "Age": [{"exists": true}]
}
```

次の例は、Location が日本で、Browser が Safari または Device がタブレットの場合に一致します。

```
{
 "Location": ["Japan"],
 "$or": [
   {"Browser": ["Safari"]},
   {"Device": ["Tablet"]}
 ]
}
```

セグメントの作成

セグメントを作成すると、それをあらゆるプロジェクトのあらゆる起動や実験で使用できます。

セグメントを作成するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで、[Application Signals]、[Evidently] の順に選択します。
3. [Segments] (セグメント) タブを選択します。
4. [Create segment] を選択します。
5. [Segment name] (セグメント名) に、このセグメントを識別するために使用する名前を入力します。

   必要に応じて説明を追加することもできます。
6. [Segment pattern] (セグメントパターン) に、ルールパターンを定義する JSON ブロックを入力します。ルールパターンの構文の詳細については、「セグメントルールのパターンの構文」を参照してください。

起動の作成

新しい機能や変更を特定の割合のユーザーに公開するには、起動を作成します。そうすると、すべてのユーザーに機能をロールアウトする前に、ページのロード時間やコンバージョンなどの主要なメトリクスをモニターリングできます。

起動を追加する前に、プロジェクトを作成しておく必要があります。詳細については、「 の新規プロジェクトの作成」を参照してください。

起動を追加するときは、すでに作成した機能を使用するか、起動の作成中に新しい機能を作成できます。

プロジェクトに起動を追加するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで、[Application Signals]、[Evidently] の順に選択します。
3. プロジェクトの名前の横にあるボタンをクリックし、[Project actions] (プロジェクトアクション)、[Create launch] (起動を作成) を選択します。
4. [Launch name] (起動名) に、このプロジェクト内でこの機能を識別するために使用する名前を入力します。

   必要に応じて説明を追加することもできます。
5. [Select from existing features] (既存の機能から選択) または [Add new feature] (新機能を追加) のいずれかを選択します。

   既存の機能を使用する場合は、その機能を [Feature name] (機能名) で選択します。

   [Add new feature] (新機能を追加) を選択した場合は、次の作業を行います。

   a. [Feature name] (機能名) に、このプロジェクト内でこの機能を識別するために使用する名前を入力します。必要に応じて説明を追加することもできます。

   b. [Feature variations] (機能のバリエーション) では、[Variation type] (バリエーションの型) として [Boolean]、[Long]、[Double]、または [String] を選択します。詳細については、「バリエーションの型」を参照してください。

   c. 機能に最大 5 つのバリエーションを追加します。各バリエーションの [Value] (値) は、選択した Variation type (バリエーションの型) として有効である必要があります。

      デフォルトにするバリエーションの 1 つを指定します。これは、他のバリエーションが比較対象とするベースラインであり、現在ユーザーに提供されているバリエーションである

ことが望ましいです。実験を停止すると、このデフォルトのバリエーションがすべてのユーザーに提供されます。

d. [Sample code] (サンプルコード) を選択します。コード例は、バリエーションをセットアップしてそこにユーザーセッションを割り当てるためにアプリケーションに追加する必要があるものを示しています。コードには JavaScript、Java、および Python を選択できます。

今すぐアプリケーションにコードを追加する必要はありませんが、起動を開始する前に追加する必要があります。

詳細については、「アプリケーションにコードを追加する」を参照してください。

6. [Launch configuration] (起動設定) で、起動をすぐに開始するか、後で開始するようにスケジュールするかを選択します。

7. (オプション) 一般的なオーディエンスに使用するトラフィック分割ではなく、定義したオーディエンスセグメントに異なるトラフィック分割を指定するには、[Add Segment Overrides] (セグメントオーバーライドを追加) を選択します。

[Segment Overrides] (セグメントオーバーライド) で、セグメントを選択し、そのセグメントに使用するトラフィック分割を定義します。

必要に応じて、[Add Segment Override] (セグメントオーバーライドを追加) を選択して、トラフィック分割を定義するセグメントをさらに定義できます。起動には最大 6 つのセグメントオーバーライドを含めることができます。

詳細については、「セグメントを使用してオーディエンスを絞り込む」を参照してください。

8. [Traffic configuration] (トラフィック設定) で、セグメントオーバーライドと一致しない一般のオーディエンスの各バリエーションに割り当てるトラフィックの割合を選択します。また、バリエーションをユーザーへの提供から除外することもできます。

[Traffic summary] (トラフィックの概要) は、全体の中でこの起動に利用可能なトラフィックの量を示します。

9. 後で開始するように起動をスケジュールする場合は、複数のステップを起動に追加できます。各ステップでは、バリエーションの提供に異なる割合を使用できます。これを行うには、[Add another step] (別のステップを追加) を選択してから、次のステップのスケジュールとトラフィックの割合を指定します。1 つの起動には、最大 5 つのステップを含めることができます。

10. 起動中にメトリクスを使用して機能のパフォーマンスを追跡する場合は、[Metrics] (メトリクス)、[Add metric] (メトリクスを追加) を選択します。CloudWatch RUM メトリクスまたはカスタムメトリクスのいずれかを使用できます。

カスタムメトリクスを使用する場合、そのメトリクスを Amazon EventBridge ルールで作成できます。カスタムメトリクスを作成するには、次の操作を行います。

- [Custom metrics] (カスタムメトリクス) を選択し、メトリクスの名前を入力します。
- [Metric rule] (メトリクスルール) で、[Entity ID] (エンティティ ID) にエンティティを識別する方法を入力します。これは、記録されるメトリクス値を発生するアクションを実行するユーザーまたはセッションです。例は `userDetails.userID` です。
- [Value key] (値キー) にメトリクスを生成するために追跡する値を入力します。
- 必要に応じて、メトリクスの単位の名前を入力します。この単位名は表示専用で、Evidently コンソールのグラフで使用します。

これらのフィールドを入力すると、EventBridge ルールをコーディングしてメトリクスを作成する方法の例がボックスに表示されます。EventBridge の詳細については、「Amazon EventBridge とは」を参照してください。

RUM メトリクスを使用するには、アプリケーションの RUM アプリケーションモニターが既にセットアップされている必要があります。詳細については、「CloudWatch RUM を使用するためにアプリケーションをセットアップする」を参照してください。

> **Note**
>
> RUM メトリクスを使用し、アプリケーションモニターがユーザーセッションの 100% をサンプリングするように設定されていない場合、起動に参加するすべてのユーザーセッションが Evidently にメトリクスを送信するわけではありません。起動メトリクスが確実に正確であるようにするために、アプリケーションモニターはサンプリングにユーザーセッションの 100% を使用することをお勧めします。

11. (オプション) 起動に少なくとも 1 つのメトリクスを作成する場合、既存の CloudWatch アラームをこの起動に関連付けることができます。これを行うには、[Associate CloudWatch alarms] (CloudWatch アラームの関連付け) を選択します。

    アラームを起動に関連付けると、CloudWatch Evidently はプロジェクト名と起動名を使用してタグをアラームに追加する必要があります。これは、CloudWatch Evidently がコンソールの起動情報に正しいアラームを表示できるようにするためです。

    CloudWatch Evidently がこれらのタグを追加することを確認するには、[Allow Evidently to tag the alarm resource identified below with this launch resource.] (この起動リソースを使用

して Evidently が以下に示すアラームリソースにタグを付けることを許可) を選択します。次に、[Associate alarm] (アラームの関連付け) を選択し、アラーム名を入力します。

CloudWatch アラームの作成については、「Amazon CloudWatch でのアラームの使用」を参照してください。

12. (オプション) この起動にタグを追加するには、[Tags] (タグ)、[Add new tag] (新しいタグを追加) を選択します。

    [Key] (キー) に、タグの名前を入力します。[値] では、任意でタグに値を追加できます。

    (オプション) 別のタグを追加するには、[Add new tag] (新しいタグを追加) を再度選択します。

    詳細については、「AWS リソースのタグ付け」を参照してください。
13. [Create launch] (起動を作成) を選択します。

実験の作成

実験を使用して、異なるバージョンの機能またはウェブサイトをテストし、実際のユーザーセッションからデータを収集します。この方法で、証拠とデータに基づいてアプリケーションでの選択ができます。

実験を追加する前に、プロジェクトを作成しておく必要があります。詳細については、「 の新規プロジェクトの作成」を参照してください。

実験を追加するときは、既に作成した機能を使用するか、実験の作成中に新しい機能を作成できます。

プロジェクトに実験を追加するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで、[Application Signals]、[Evidently] の順に選択します。
3. プロジェクトの名前の横にあるボタンをクリックし、[Project actions] (プロジェクトアクション)、[Create experiment] (実験を作成) を選択します。
4. [Experiment name] (実験名) に、このプロジェクト内でこの機能を識別するために使用する名前を入力します。

    必要に応じて説明を追加することもできます。
5. [Select from existing features] (既存の機能から選択) または [Add new feature] (新機能を追加) のいずれかを選択します。

既存の機能を使用する場合は、その機能を [Feature name] (機能名) で選択します。

[Add new feature] (新機能を追加) を選択した場合は、次の作業を行います。

a. [Feature name] (機能名) に、このプロジェクト内でこの機能を識別するために使用する名前を入力します。必要に応じて説明を入力することもできます。

b. [Feature variations] (機能のバリエーション) では、[Variation type] (バリエーションの型) として [Boolean]、[Long]、[Double]、または [String] を選択します。型では、各バリエーションに使用される値の型を定義します。詳細については、「バリエーションの型」を参照してください。

c. 機能に最大 5 つのバリエーションを追加します。各バリエーションの [Value] (値) は、選択した Variation type (バリエーションの型) として有効である必要があります。

   デフォルトにするバリエーションの 1 つを指定します。これは、他のバリエーションが比較対象とするベースラインであり、現在ユーザーに提供されているバリエーションであることが望ましいです。この機能を使用する実験を停止すると、デフォルトのバリエーションが以前に実験に参加していた特定の割合のユーザーに提供されます。

d. [Sample code] (サンプルコード) を選択します。コード例は、バリエーションをセットアップしてそこにユーザーセッションを割り当てるためにアプリケーションに追加する必要があるものを示しています。コードには JavaScript、Java、および Python を選択できます。

   今すぐアプリケーションにコードを追加する必要はありませんが、実験を開始する前に追加する必要があります。詳細については、「アプリケーションにコードを追加する」を参照してください。

6. この実験をそのセグメントに適合するユーザーのみに適用する場合は、必要に応じて [Audience] (オーディエンス) で作成したセグメントを選択します。セグメントの詳細については、「セグメントを使用してオーディエンスを絞り込む」を参照してください。

7. [Traffic split for the experiment] (実験のトラフィック分割) で、実験でセッションを使用することを選択したオーディエンスの割合を指定します。次に、実験で使用するさまざまなバリエーションにトラフィックを割り当てます。

   起動と実験の両方が同じ機能で同時に実行されている場合、対象者に対して最初に起動に関する振り分けが行われます。そして、対象者全体の中から起動用に指定された割合のトラフィックが取得されます。その後、ここで指定する割合は、残りのオーディエンスの中から実験に使用される割合になります。さらに残ったトラフィックは、デフォルトのバリエーションで提供されます。

8. [Metrics] (メトリクス) で、実験中のバリエーションを評価するために使用するメトリクスを選択します。評価には、少なくとも 1 つのメトリクスを使用する必要があります。

   a. [Metric source] (メトリクスのソース) で、CloudWatch RUM メトリクスを使用するか、カスタムメトリクスを使用するかを選択します。

   b. メトリクスの名前を入力します。[Goal] (目標) で、メトリクスの値が大きい方が良いバリエーションである場合、[Increase] (増加) を選択します。メトリクスの値が小さい方が良いバリエーションである場合、[Decrease] (減少) を選択します。

   c. カスタムメトリクスを使用している場合は、Amazon EventBridge ルールを使用してメトリクスを作成できます。カスタムメトリクスを作成するには、次の操作を行います。

      - [Metric rule] (メトリクスルール) で、[Entity ID] (エンティティ ID) に、エンティティを識別する方法を入力します。これは、記録されるメトリクス値を発生するアクションを実行するユーザーまたはセッションです。例は `userDetails.userID` です。
      - [Value key] (値キー) にメトリクスを生成するために追跡する値を入力します。
      - 必要に応じて、メトリクスの単位の名前を入力します。この単位名は表示専用で、Evidently コンソールのグラフで使用します。

      RUM メトリクスは、このアプリケーションをモニターリングするように RUM をセットアップしている場合にのみ使用できます。詳細については、「CloudWatch RUM を使用する」を参照してください。

      > **Note**
      >
      > RUM メトリクスを使用し、アプリケーションモニターがユーザーセッションの 100% をサンプリングするように設定されていない場合、実験のすべてのユーザーセッションがメトリクスを Evidently に送信するわけではありません。実験メトリクスが確実に正確であるようにするために、アプリケーションモニターはサンプリングにユーザーセッションの 100% を使用することをお勧めします。

   d. (オプション) 評価するメトリクスをさらに追加するには、[Add metric] (メトリクスを追加) を選択します。実験中に 3 つのメトリクスを評価できます。

9. (オプション) この実験で使用する CloudWatch アラームを作成するには、[CloudWatch alarms] (CloudWatch アラーム) を選択します。アラームで、各バリエーションとデフォルトのバリエーションの結果の差が指定したしきい値より大きいかどうかをモニターリングできます。バリエー

ションのパフォーマンスがデフォルトのバリエーションよりも悪く、その差がしきい値より大きい場合、アラーム状態になり、通知されます。

ここでアラームを作成すると、デフォルトのバリエーションではないバリエーションごとに 1 つずつアラームが作成されます。

アラームを作成する場合は、以下のように指定します。

- [Metric name] (メトリクス名) で、アラームに使用する実験メトリクスを選択します。
- [Alarm condition] (アラーム条件) で、バリエーションのメトリクス値をデフォルトバリエーションのメトリクス値と比較した場合に、アラームがアラーム状態になる条件を選択します。例えば、数字が大きいとバリエーションのパフォーマンスが悪いことを示している場合、[Greater] (より大きい) または [Greater/Equal] (以上) を選択します。これは、メトリクスがページのロード時間を測定している場合などに適しています。
- しきい値の数値を入力します。これは、アラームが ALARM 状態になるときのパフォーマンスの差のパーセンテージです。
- [Average over period] (期間の平均値) で、比較する前に、各バリエーションのメトリクスデータをまとめて集計する量を選択します。

[Add new alarm] (新しいアラームを追加) を再度選択して、実験にさらにアラームを追加できます。

次に [Set notifications for the alarm] (アラームの通知を設定) を選択して、アラーム通知を送信する Amazon Simple Notification Service トピックを選択または作成します。詳細については、「Amazon SNS 通知の設定」を参照してください。

10. (オプション) この実験にタグを追加するには、[Tags] (タグ)、[Add new tag] (新しいタグを追加) を選択します。

    [Key] (キー) に、タグの名前を入力します。[値] では、任意でタグに値を追加できます。

    (オプション) 別のタグを追加するには、[Add new tag] (新しいタグを追加) を再度選択します。

    詳細については、「AWS リソースのタグ付け」を参照してください。
11. [Create experiment (実験の作成)] を選択します。
12. 機能バリアントをまだ作成していない場合は、アプリケーションに作成します。
13. [完了] をクリックします。実験は開始するまで開始されません。

以下の手順を完了すると、実験は直ちに開始されます。

作成した実験を開始するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで、[Application Signals]、[Evidently] の順に選択します。
3. プロジェクトの名前を選択します。
4. [Experiments] (実験) タブを選択します。
5. 実験の名前の横にあるボタンをクリックし、[Actions] (アクション)、[Start experiment] (実験を開始) を選択します。
6. (オプション) 作成時に行った実験設定を表示または変更するには、[Experiment setup] (実験の設定) を選択します。
7. 実験が終了する時間を選択します。
8. [Start experiment] (実験を開始) を選択します。

   実験がすぐに開始されます。

機能、起動、実験を管理する

作成した機能、起動、および実験を管理するには、これらのセクションの手順を使用します。

トピック

機能の現在の評価ルールと対象者トラフィックを確認する

CloudWatch Evidently コンソールを使用して、機能の評価ルールが機能の現在の起動、実験、およびバリエーションに対象者トラフィックをどのように割り当てているかを確認できます。

機能の対象者トラフィックを表示するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで、[Application Signals]、[Evidently] の順に選択します。
3. 機能を含むプロジェクトの名前を選択します。
4. [Features] (機能) タブを選択します。
5. 機能の名前を選択します。

   [Evaluation rules] (評価ルール) タブでは、機能の対象者トラフィックのフローを次のように確認できます。

   - まず、オーバーライドが評価されます。これらは、特定のユーザーに常に特定のバリエーションを提供することを指定します。オーバーライドが割り当てられているユーザーのセッションは、起動または実験メトリクスに寄与しません。
   - 次に、残りのトラフィックは、進行中の起動があれば、そこで利用可能になります。起動が進行中の場合は、[Launches] (起動) セクションには、起動名と機能のバリエーション間で分割された起動トラフィックが表示されます。[Launches] (起動) セクションの右側の [Traffic] (トラフィック) インジケータには、この起動に割り当てられている利用可能な対象者 (オーバーライド後) の量が表示されます。起動に割り当てられていない残りのトラフィックは、実験があればそこに流れ、さらにデフォルトのバリエーションに流れます。
   - 次に、残りのトラフィックが、進行中の実験があればそこで利用可能になります。実験が進行中の場合は、[Experiments] (実験) セクションには、実験の名前と進行状況が表示されます。[Experiments] (実験) セクションの右側の [Traffic] (トラフィック) インジケータには、この実験に割り当てられている利用可能な対象者 (オーバーライドおよび起動後) の量が表示されます。起動または実験に割り当てられていない残りのトラフィックは、機能のデフォルトのバリエーションとして提供されます。

起動トラフィックを変更する

起動へのトラフィック割り当ては、起動の進行中を含め、いつでも変更できます。

同じ機能に対して進行中の起動と進行中の実験の両方がある場合、機能トラフィックを変更すると、実験トラフィックが変更されます。実験で利用可能な対象者は、対象者全体の中で起動に割り当てられていない部分であるためです。起動トラフィックを増やすと、実験に利用できる対象者が減少し、起動トラフィックを減らしたり、起動を終了したりすると、実験に利用できる対象者が増加します。

起動へのトラフィック割り当てを変更するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで、[Application Signals]、[Evidently] の順に選択します。
3. 起動を含むプロジェクトの名前を選択します。
4. [Launches] (起動) タブを選択します。
5. 起動名を選択します。

   [Modify launch traffic] (起動トラフィックを変更) を選択します。
6. [Serve] (提供) で、各バリエーションに割り当てる新しいトラフィックの割合を選択します。また、バリエーションをユーザーへの提供から除外することもできます。これらの値を変更すると、機能トラフィック全体に対する更新の影響が [Traffic summary] (トラフィックの概要) で確認できます。

   [Traffic summary] (トラフィックの概要) に、この起動で使用できるトラフィックの総量と、その使用できるトラフィックの中でこの起動に割り当てられている量が表示されます。
7. Modify を選択します。

起動の今後のステップを変更する

まだ実行されていない起動ステップの設定を変更したり、起動にステップを追加したりできます。

起動のステップを変更するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで、[Application Signals]、[Evidently] の順に選択します。
3. 起動を含むプロジェクトの名前を選択します。
4. [Launches] (起動) タブを選択します。
5. 起動名を選択します。

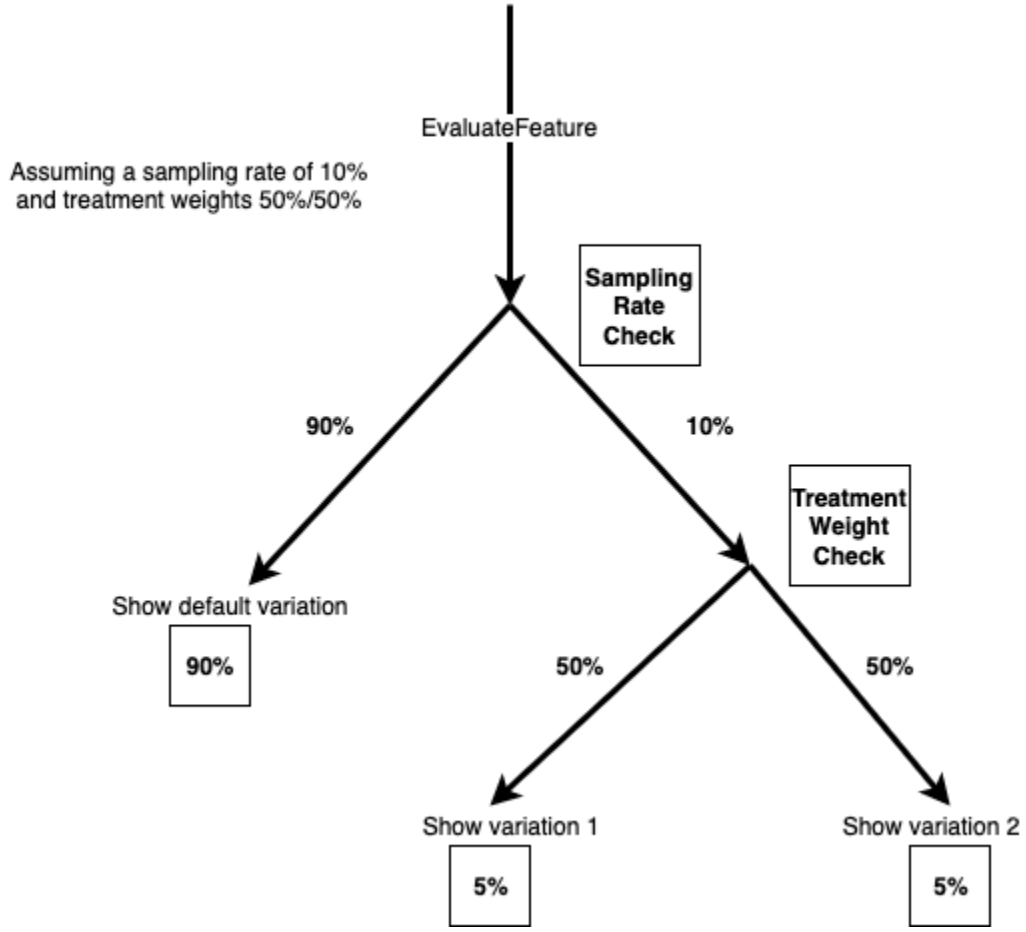
   [Modify launch traffic] (起動トラフィックを変更) を選択します。
6. [Schedule launch] (起動のスケジュールを設定) を選択します。
7. まだ開始していない手順については、実験で使用可能な対象者の割合を変更できます。また、バリエーション間でトラフィックの割り当て方法を変更することもできます。

起動にステップを追加するには、[Add another step] (別のステップを追加) を選択します。起動は、最大 5 つのステップを持つことができます。

8. Modify を選択します。

実験トラフィックを変更する

実験のサンプリングレートはいつでも変更でき、実験の進行中に変更することも可能です。ただし、実験の実行後に処理の重みを更新することはできません。そのため、実験の実行後に実験に公開されるトラフィックの合計を変更することはできても、各処理に対する相対的な割り当てを変更することはできません。進行中の実験のトラフィックを変更する場合は、バイアスを発生させないように、トラフィック割り当てを増やすだけにすることをお勧めします。



実験のトラフィック割り当てを変更するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。

2. ナビゲーションペインで [Application monitoring] (アプリケーションのモニターリング)、[Evidently] を選択します。
3. 起動を含むプロジェクトの名前を選択します。
4. [Experiments] (実験) タブを選択します。
5. 起動名を選択します。
6. [Modify experiment traffic] (実験トラフィックを変更) を選択します。
7. パーセンテージを入力するか、スライダーを使用して、この実験に割り当てる使用可能なトラフィックの量を指定します。使用可能なトラフィックは、現在の起動に割り当てられているトラフィックがある場合、総対象者からそれを引いた値です。起動または実験に割り当てられていないトラフィックは、デフォルトのバリエーションで提供されます。
8. Modify を選択します。

起動を停止する

進行中の起動を停止すると、その起動を再開したり、再スタートしたりできなくなります。また、トラフィック割り当てのルールとして評価されなくなり、その起動に割り当てられていたトラフィックは、機能の実験がある場合、そこで利用可能になります。機能の実験がない場合、起動が停止した後、すべてのトラフィックがデフォルトのバリエーションで配信されます。

起動を恒久的に停止するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで、[Application Signals]、[Evidently] の順に選択します。
3. 起動を含むプロジェクトの名前を選択します。
4. [Launch] (起動) タブを選択します。
5. 起動の名前の左にあるボタンをクリックします。
6. [Actions] (アクション)、[Cancel launch] (起動をキャンセル) の順に、または [Actions] (アクション)、[Mark as complete] (完了としてマーク) の順に選択します。

実験を停止する

進行中の実験を停止すると、その実験を再開したり、再スタートしたりできなくなります。それまで実験に使用されていた部分のトラフィックは、デフォルトのバリエーションで提供されるようになります。

実験を手動で停止せず、終了日を過ぎた場合は、トラフィックは変化しません。実験に割り当てられていた部分のトラフィックは、そのまま実験に送られます。これを停止し、代わりに実験トラフィックがデフォルトのバリエーションに提供されるようにするには、実験を完了としてマークします。

実験を停止する場合、キャンセルするか、完了としてマークするかを選択できます。キャンセルすると、実験のリストに [Cancelled] (キャンセル済み) と表示されます。完了としてマークすると、[Completed] (完了) と表示されます。

実験を恒久的に停止するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで、[Application Signals]、[Evidently] の順に選択します。
3. 実験を含むプロジェクトの名前を選択します。
4. [Experiments] (実験) タブを選択します。
5. 実験の名前の左にあるボタンをクリックします。
6. [Actions] (アクション)、[Cancel experiment] (実験をキャンセル) の順に、または [Actions] (アクション)、[Mark as complete] (完了としてマーク) の順に選択します。

アプリケーションにコードを追加する

CloudWatch Evidently を使用するには、アプリケーションにコードを追加して、各ユーザーセッションにバリエーションを割り当て、Evidently にメトリクスを送信します。CloudWatch Evidently `EvaluateFeature` オペレーションを使用してユーザーセッションにバリエーションを割り当てます。`PutProjectEvents` オペレーションを使用してイベントを送信し、起動または実験のメトリクスを計算するために使用します。

バリエーションまたはカスタムメトリクスを作成すると、CloudWatch Evidently コンソールに追加する必要があるコードのサンプルが表示されます。

エンドツーエンドの例については、「チュートリアル: Evidently のサンプルアプリケーションを使用した A/B テスト」を参照してください。

EvaluateFeature の使用

起動または実験で機能のバリエーションが使用される場合、アプリケーションは EvaluateFeature オペレーションを使用して、各ユーザーセッションにバリエーションを割り当てます。ユーザーへのバリエーションの割り当ては、評価イベントです。このオペレーションを呼び出すとき、以下を渡します。

- 機能名 – 必須。Evidently では、起動または実験の機能評価ルールに従って評価が処理され、エンティティのバリエーションが選択されます。
- entityId – 必須。一意のユーザーを表します。
- evaluationContext – オプション。ユーザーに関する追加情報を表す JSON オブジェクト。セグメントを作成した場合、Evidently では機能評価中にこの値を使用して、ユーザーをオーディエンスのセグメントと照合します。詳細については、「セグメントを使用してオーディエンスを絞り込む」を参照してください。

  Evidently に送信できる `evaluationContext` の値の例を次に示します。

```
{
    "Browser": "Chrome",
    "Location": {
        "Country": "United States",
        "Zipcode": 98007
    }
}
```

Sticky 評価

CloudWatch は明らかに「スティッキー」評価を使用します。`entityId` の単一の構成、機能、機能の構成および `evaluationContext` は、常に同じバリエーション割り当てを受け取ります。この変動の割り当てが変更されるのは、エンティティがオーバーライドに追加されるか、実験トラフィックがダイヤルアップされる場合のみです。

機能の構成には、以下が含まれます。

- 機能のバリエーション
- この機能で現在実行中の実験のバリエーションの構成 (各バリエーションに割り当てられたパーセンテージ) (ある場合)。
- この機能で現在実行中のローンチのバリエーションの設定 (ある場合)。バリエーションの構成には、定義済みのセグメントオーバーライドが含まれます (ある場合)。

実験のトラフィック配分を増やしても、以前に実験処理グループに割り当てられていた `entityId` には、引き続き同じ処理が行われます。以前にコントロールグループに割り当てられていた `entityId` は、実験のために指定されたバリエーションの構成に従って、実験処理グループに割り当てられている場合があります。

実験のトラフィック配分を減少させると、entityId が処理グループからコントロールグループに移行する可能性はありますが、異なる処理グループに移行することはありません。

PutProjectEvents の使用

明らかにカスタムメトリクスをコーディングするには、 PutProjectEvents オペレーションを使用します。ペイロードの簡単な例を次に示します。

```
{
    "events": [
        {
            "timestamp": {{$timestamp}},
            "type": "aws.evidently.custom",
            "data": "{\"details\": {\"pageLoadTime\": 800.0}, \"userDetails\":
 {\"userId\": \"test-user\"}}"
        }
    ]
}
```

entityIdKey は entityId のままにすることも、名前を変更して userId など他のものに変更することもできます。実際のイベントでは、entityId は、ユーザー名、セッション ID などになります。

```
"metricDefinition":{
    "name": "noFilter",
    "entityIdKey": "userDetails.userId", //should be consistent with jsonValue in
 events "data" fields
    "valueKey": "details.pageLoadTime"
},
```

イベントが正しい起動または実験に関連付けられていることを確認するには、同じことを渡す必要があります。EvaluateFeature と PutProjectEvents の両方を呼び出すときは、同じ entityId を渡す必要があります。EvaluateFeature コールの後に必ず PutProjectEvents を呼び出してください。そうしないと、データが削除され、CloudWatch では明らかに使用されません。

PutProjectEvents オペレーションでは、入力パラメータとしてフィーチャ名は不要です。このように、単一のイベントを複数の実験で使用できます。例えば、 entityId を userDetails.userId に設定して EvaluateFeature を呼び出すとします。2 つ以上の実験を実行している場合、そのユーザーのセッションから 1 つのイベントでそれらの各実験に対

してメトリクスを放出できます。これを行うには、同じ entityId を使用して、実験ごとに1回PutProjectEvents を呼び出します。

[Timing] (タイミング)

アプリケーションが EvaluateFeature をコールした後では、PutProjectEvents からのメトリクイベントがその評価に基づいて帰属される1時間の期間があります。1 時間後にさらにイベントが発生した場合、そのイベントは帰属しません。

ただし、その最初の呼び出しの 1 時間のウィンドウ中に同じ entityId が新しい EvaluateFeature 呼び出しに使用された場合、代わりに後の EvaluateFeature 結果が使用され、1 時間のタイマーが再開されます。これは、前の Sticky 評価セクションでの説明のように、実験トラフィックが 2 つの割り当て間でダイヤルアップされる場合など、特定の状況でのみ発生します。

エンドツーエンドの例については、「チュートリアル: Evidently のサンプルアプリケーションを使用した A/B テスト」を参照してください。

プロジェクトデータストレージ

Evidently は次の 2 つのタイプのイベントを収集します。

- 評価イベントは、ユーザーセッションに割り当てられる機能のバリエーションに関連しています。Evidently は、これらのイベントを使用して、メトリクスやその他の実験および起動データを生成します。これらは、Evidently コンソールで表示できます。

  また、これらの評価イベントを Amazon CloudWatch Logs または Amazon S3 に保存することもできます。
- カスタムイベントは、クリックやチェックアウトなどのユーザーアクションからメトリクスを生成するために使用されます。Evidently では、カスタムイベントを保存するためのメソッドは提供されていません。保存する場合は、アプリケーションコードを変更して、Evidently 以外のストレージを選択して送信する必要があります。

評価イベントログの形式

CloudWatch Logs または Amazon S3 に評価イベントを保存する場合、各評価イベントは次の形式でログイベントとして保存されます。

```
{
```

```
    "event_timestamp": 1642624900215,
    "event_type": "evaluation",
    "version": "1.0.0",
    "project_arn": "arn:aws:evidently:us-east-1:123456789012:project/petfood",
    "feature": "petfood-upsell-text",
    "variation": "Variation1",
    "entity_id": "7",
    "entity_attributes": {},
    "evaluation_type": "EXPERIMENT_RULE_MATCH",
    "treatment": "Variation1",
    "experiment": "petfood-experiment-2"
}
```

上記の評価イベント形式の詳細を以下に示します。

- タイムスタンプは UNIX 時間 (ミリ秒) です。
- バリエーションは、このユーザーセッションに割り当てられている特徴のバリエーションの名前です。
- エンティティ ID は文字列です。
- エンティティ属性は、クライアントから送信された任意の値のハッシュです。たとえば、entityId が青または緑にマップされている場合、必要に応じて UserID、セッションデータ、または相関関係とデータウェアハウスの観点から目的のものを送信できます。

Amazon S3 での評価イベントストレージの IAM ポリシーと暗号化

Amazon S3 を使用して評価イベントを保存する場合、次のような IAM ポリシーを追加して、Evidently が Amazon S3 バケットにログを公開できるようにする必要があります。これは、Amazon S3 バケットとそれに含まれているオブジェクトはプライベートであり、デフォルトで他のサービスへのアクセスを許可しないためです。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AWSLogDeliveryWrite",
            "Effect": "Allow",
            "Principal": {"Service": "delivery.logs.amazonaws.com"},
            "Action": "s3:PutObject",
            "Resource": "arn:aws:s3:::bucket_name/optional_folder/AWSLogs/account_id/
*",
```

```
            "Condition": {"StringEquals": {"s3:x-amz-acl": "bucket-owner-full-
control"}}
        },
        {
            "Sid": "AWSLogDeliveryCheck",
            "Effect": "Allow",
            "Principal": {"Service": "delivery.logs.amazonaws.com"},
            "Action": ["s3:GetBucketAcl", "s3:ListBucket"],
            "Resource": "arn:aws:s3:::bucket_name"
        }
    ]
}
```

Evidently データを Amazon S3 に保存する場合は、AWS Key Management Service キーを使用したサーバー側の暗号化 (SSE-KMS) を使用して暗号化することもできます。詳細については、「サーバー側の暗号化を使用したデータの保護」を参照してください。

AWS KMS からのカスタマーマネージドキーを使用する場合は、キーの IAM ポリシーに以下を追加する必要があります。これにより、Evidently はバケットに書き込むことができます。

```
{
    "Sid": "AllowEvidentlyToUseCustomerManagedKey",
    "Effect": "Allow",
    "Principal": {
        "Service": [
            "delivery.logs.amazonaws.com"
        ]
    },
   "Action": [
       "kms:Encrypt",
       "kms:Decrypt",
       "kms:ReEncrypt*",
       "kms:GenerateDataKey*",
       "kms:DescribeKey"
    ],
    "Resource": "*"
}
```

Evidently の結果算出方法

Amazon CloudWatch Evidently の A/B テストを、データドリブンな意思決定のツールとして使用できます。A/B テストでは、ユーザーはコントロールグループ (デフォルトのバリエーションとも呼ば

れる) と処理グループ (テストされたバリエーションとも呼ばれる) のいずれかにランダムに割り当てられます。例えば、コントロールグループのユーザーは、実験開始前と同じようにウェブサイト、サービス、またはアプリケーションを体験する可能性があります。一方、処理グループのユーザーもこの変化を体験する可能性があります。

CloudWatch Evidently では、1 つの実験で最大 5 つの異なるバリエーションがサポートされています。Evidently がこれらのバリエーションにトラフィックをランダムに割り当てます。これにより、各グループのビジネスメトリクス (収益など) やパフォーマンスメトリクス (レイテンシーなど) を追跡できます。Evidently は以下の操作を行います。

- コントロールグループと処理グループを比較します。(例えば、新しいチェックアウトプロセスで収益が増加するか減少するかを比較します。)
- 処理グループとコントロールグループで観察された差が有意であるかどうかを示します。これについて、Evidently には頻度論的有意水準およびベイズ確率という 2 つのアプローチがあります。

頻度論的アプローチとベイズアプローチを使用する理由

処理グループをコントロールグループと比較しても効果がない場合や、処理グループがコントロールグループと同一である場合を考えてみましょう (A/A テスト)。それでも、処理グループとコントロールグループとの間にわずかなデータの差が見られます。これは、テスト参加者がウェブサイト、サービス、またはアプリケーションの全ユーザーのごく一部を占める限定されたユーザーサンプルで構成されているためです。頻度論的有意水準とベイズ確率を使用することで、観測された差が有意であるか、それとも偶然によるものなのかを知ることができます。

Evidently は次の点を考慮して、観測された差が有意であるかどうかを判断します。

- 差はどの程度か
- テストに含まれるサンプル数
- データの配布方法

Evidently での頻度論的分析

Evidently は逐次テストを使用するため、頻度論的統計の一般的な落とし穴であるピーキングの問題を回避することができます。ピーキングとは、進行中の A/B テストの結果をチェックして、テストを停止し、観察された結果に基づいて意思決定を行うことです。逐次テストの詳細については、Howard (共著) の「Time-uniform, nonparametric, nonasymptotic confidence sequences」(時間

一様、非パラメトリック、非漸近信頼系列) (Ann. Statist. 49 (2) 1055 - 1080, 2021)を参照してください。

Evidently の結果は常に有効 (常に有効な結果) であるため、実験中に結果を覗いたとしても、確かな結論を導き出すことができます。そのため、結果に既に有意な意味がある場合は、予定時間前に実験を中止することができるため、実験にかかるコストの一部を削減することができます。

Evidently は、ターゲットメトリクスのテストされたバリエーションとデフォルトのバリエーションとの差について、常に有効な有意水準と常に有効な 95% 信頼区間を生成します。実験の [Result] (結果) の列には、テストされたバリエーションのパフォーマンスが表示され、次のいずれかになります。

- 非決定的 - 有意水準が 95% 未満である
- 良い - 有意水準が 95% 以上であり、次のいずれかに該当する
  - 95% 信頼区間の下限が 0 より大きく、メトリクスが増加する
  - 95% 信頼区間の上限が 0 より低く、メトリクスが減少する
- 悪い — 有意水準が 95% 以上で、次のいずれかに該当する
  - 95% 信頼区間の上限が 0 より大きく、メトリクスが増加する
  - 95% 信頼区間の下限は 0 より低く、メトリクスが減少する
- 最良 — 実験には、デフォルトのバリエーションに加えて 2 つ以上のテスト済みのバリエーションがあり、次の条件が満たされています。
  - このバリエーションは「良い」に該当します
  - 以下のいずれかに該当します:
    - 95% 信頼区間の下限は、他のすべてのバリエーションの 95% 信頼区間の上限よりも高く、メトリクスは増加します
    - 95% 信頼区間の上限は、他のすべてのバリエーションの 95% 信頼区間の下限よりも低く、メトリクスは減少します

Evidently でのベイズ分析

ベイズ分析では、テスト済みのバリエーションの平均がデフォルトのバリエーションの平均よりも大きいか小さいかの確率を計算することができます。Evidently は、共役事前分布を用いてターゲットメトリクスの平均値に対してベイズ推定を行います。共役事前分布を使用すると、Evidenly はベイズ分析に必要な事後分布をより効率的に推測できます。

Evidently は実験終了日まで待機してから、ベイズ分析の結果を計算します。結果ページには次の情報が表示されます。

- 増加確率 - テスト済みのバリエーションにおけるメトリクスの平均が、デフォルトのバリエーションの平均よりも少なくとも 3% 大きい確率
- 減少確率 — テスト済みのバリエーションにおけるメトリクスの平均が、デフォルトのバリエーションの平均よりも少なくとも 3% 小さい確率
- 変化がない確率 — テスト済みのバリエーションにおけるメトリクスの平均がデフォルトのバリエーションの平均の ±3% 以内にある確率

[Result] (結果) の列には、バリエーションのパフォーマンスが示され、次のいずれかに該当します。

- 良い — 増加確率が 90% 以上でメトリクスが増加するか、または減少確率が 90% 以上でメトリクスが減少する
- 悪い — 減少確率が 90% 以上でメトリクスが増加するか、または増加確率が 90% 以上でメトリクスが減少する

ダッシュボードで起動結果を表示する

実験の進行中および完了後に、実験の進行状況とメトリクスの結果を確認できます。

起動の進捗状況と結果を確認するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで、[Application Signals]、[Evidently] の順に選択します。
3. 起動を含むプロジェクトの名前を選択します。
4. [Launch] (起動) タブを選択します。
5. 起動名を選択します。
6. 各ステップの起動手順とトラフィック割り当てを確認するには、[Launch] (起動) タブを選択します。
7. 各バリエーションに割り当てられたユーザーセッションの数を時間の経過とともに確認し、起動での各バリエーションのパフォーマンスメトリクスを表示するには、[Monitoring] (モニターリング) タブを選択します。

   このビューには、起動中に起動アラームが ALARM 状態になっているかどうかも表示されます。

8. この起動のバリエーション、メトリクス、アラーム、タグを表示するには、[Configuration] (設定) タブを選択します。

ダッシュボードで実験結果を表示する

実験の進行中および完了後に、実験の統計結果を確認できます。実験結果は、実験の開始から 63 日後まで確認できます。CloudWatch データ保持ポリシーのため、それより後には利用できません。

各バリエーションに少なくとも 100 個のイベントが発生するまで、統計結果は表示されません。

Evidently では、実験の最後に追加のオフライン p 値分析が実行されます。オフライン p 値分析では、実験中に使用された p 値のどれかに統計的有意性が見られない場合に、統計的有意性を検出できます。

CloudWatch Evidently がどのようにテスト結果を計算するかについては、「Evidently の結果算出方法」を参照してください。

実験の結果を表示するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで、[Application Signals]、[Evidently] の順に選択します。
3. 実験を含むプロジェクトの名前を選択します。
4. [Experiments] (実験) タブを選択します。
5. 実験の名前を選択し、[Results] (結果) タブを選択します。
6. [Variation performance] (バリエーションのパフォーマンス) のそばに、表示する実験統計量を選択できるコントロールがあります。複数の統計を選択すると、Evidently によって各統計のグラフとテーブルが表示されます。

   各グラフと表には、これまでの実験の結果が表示されます。

   各グラフには、次の結果を表示できます。グラフの右側にあるコントロールを使用して、次の項目のうちどれが表示されるかを指定できます。

   - 各バリエーションで記録されたユーザーセッションイベントの数。
   - グラフの最上部で選択されたメトリクスの各バリエーションについての平均値。
   - 実験の統計的有意性。これにより、グラフの上部で選択したメトリクスについて、デフォルトのバリエーションと他の各バリエーションとの差が比較されます。

- 選択したメトリクスについての各バリエーションとデフォルトのバリエーションとの差の 95% 信頼区間の上限および下限。

テーブルには、行ごとに各バリエーションが表示されます。デフォルトではない各バリエーションについて、統計的に有意な結果を宣言するのに十分なデータを Evidently が受信したかどうかを表示します。また、統計値におけるバリエーションの改善が 95% の信頼水準に達しているかどうかも示します。

最後に、[Result] (結果) 列には、Evidently によって、この統計量に基づいてどのバリエーションが最適になるかのレコメンデーションが提示されるか、または結果が決定できないと示されます。

CloudWatch Evidently がデータを収集して保存する方法

Amazon CloudWatch Evidently は、お客様が実験や起動を実行できるように、プロジェクト設定に関連するデータを収集して保存します。このデータには以下が含まれています。

- プロジェクト、機能、起動、実験に関するメタデータ
- メトリクスイベント
- 評価データ

リソースメタデータは Amazon DynamoDB に保存されます。データはデフォルトで保管時に AWS 所有のキー を使用して暗号化されます。これらのキーは、複数の AWS アカウント で使用するために AWS のサービス が所有し管理する AWS KMS キーのコレクションです。お客様は、これらのキーの使用を閲覧、管理、監査することはできません。また、お客様のデータを暗号化するキーを保護するためにアクションを実行したりプログラムを変更したりする必要はありません。

詳細については、AWS Key Management Service デベロッパーガイドの「AWS 所有のキー」を参照してください。

Evidently メトリクスイベントと評価イベントは、お客様が所有する場所に直接配信されます。

転送中のデータは HTTPS で自動的に暗号化されます。このデータは、お客様が所有する場所に配信されます。

また、評価イベントを Amazon Simple Storage Service または Amazon CloudWatch Logs に保存することもできます。これらのサービスでデータを保護する方法の詳細については、「Amazon S3 の

デフォルトバケット暗号化の有効化」と「AWS KMS を使用して CloudWatch Logs のログデータを暗号化する」を参照してください。

データの取得

CloudWatch Evidently API を使用してデータを取得できます。プロジェクトデータを取得するには、GetProject または ListProjects を使用します。

機能データを取得するには、GetFeature または ListFeatures を使用します。

起動データを取得するには、GetLaunch または ListLaunches を使用します。

実験データを取得するには、GetExperiment、ListExperiments、または GetExperimentResults を使用します。

データの変更と削除

CloudWatch Evidently API を使用して、データを変更および削除できます。プロジェクトデータについては、UpdateProject または DeleteProject を使用します。

機能データについては、UpdateFeature または DeleteFeature を使用します。

起動データについては、UpdateLaunch または DeleteLaunch を使用します。

実験データについては、UpdateExperiment または DeleteExperiment を使用します。

Evidently のサービスにリンクされたロールを使用する

CloudWatch Evidently は AWS Identity and Access Management (IAM) サービスリンクロールを使用します。サービスにリンクされたロールは、Evidently に直接リンクされる一意のタイプの IAM ロールです。サービスにリンクされたロールは Evidently によって事前に定義されており、サービスがユーザーに代わって他の AWS のサービスを呼び出すために必要なすべてのアクセス許可が含まれています。

サービスにリンクされたロールを使用すると、必要なアクセス許可を手動で追加する必要がないため、Evidently の設定が簡単になります。Evidently は、サービスにリンクされたロールのアクセス許可を定義します。別に定義されている場合を除き、Evidently のみがそのロールを引き受けることができます。定義される許可には、信頼ポリシーとアクセス許可ポリシーが含まれており、そのアクセス許可ポリシーを他の IAM エンティティに添付することはできません。

サービスリンクロールを削除するには、まずその関連リソースを削除します。これにより、リソースへのアクセス許可を不用意に削除することができなくなるため、Evidently のリソースを保護することができます。

サービスにリンクされたロールをサポートする他のサービスについては、「IAM と連動する AWS のサービス」を参照し、[Service-linked roles] (サービスにリンクされたロール) の列内で [Yes] (はい) と表記されたサービスを確認してください。サービスにリンクされたロールに関するドキュメントをサービスで表示するには、[はい] リンクを選択します。

Evidently のサービスにリンクされたロールのアクセス許可

Evidently は、AWSServiceRoleForCloudWatchEvidently という名前のサービスにリンクされたロールを使用します。これにより、CloudWatch Evidently がお客様に代わって関連する AWS リソースを管理することを許可します。

サービスにリンクされたロール AWSServiceRoleForCloudWatchEvidently は、以下のサービスを信頼してロールを引き受けます。

- `CloudWatch Evidently`

AmazonCloudWatchEvidentlyServiceRolePolicy という名前のロールのアクセス許可ポリシーは、指定したリソースに対して以下のアクションを実行することを Evidently に許可します。

- アクション: Evidently シッククライアントで `appconfig:StartDeployment`、`appconfig:StopDeployment`、`appconfig:ListDeployments`、を実行します。

サービスにリンクされたロールの作成、編集、削除をIAM エンティティ (ユーザー、グループ、ロールなど) に許可するには、許可を設定する必要があります。詳細については、「IAM User Guide」(IAM ユーザーガイド) の「Service-linked role permissions」(サービスにリンクされたロールのアクセス権限) を参照してください。

Evidently のサービスにリンクされたロールを作成する

サービスにリンクされたロールを手動で作成する必要はありません。Evidently シッククライアントを AWS Management Console、AWS CLI、または AWS API で使い始めると、Evidently によってサービスにリンクされたロールが作成されます。

このサービスにリンクされたロールを削除した後で再度作成する必要が生じた場合は、同じ方法でアカウントにロールを再作成できます。Evidently シッククライアントを使い始めると、Evidence によってサービスにリンクされたロールが再作成されます。

Evidently のサービスにリンクされたロールを編集する

Evidently では、サービスリンクロール AWSServiceRoleForCloudWatchEvidently を編集できません。サービスにリンクされたロールを作成すると、多くのエンティティによってロールが参照される可能性があるため、ロール名を変更することはできません。ただし、IAM を使用したロールの説明の編集はできます。詳細については、「IAM ユーザーガイド」の「サービスにリンクされたロールの編集」を参照してください。

Evidently のサービスにリンクされたロールを削除する

サービスにリンクされたロールが必要な機能またはサービスが不要になった場合には、そのロールを削除することをお勧めします。そうすることで、積極的にモニタリングまたは保守されていない未使用のエンティティを排除できます。ただし、手動で削除する前に、サービスにリンクされたロールのリソースをクリーンアップする必要があります。シックライアントを使用している Evidently プロジェクトはすべて削除する必要があります。

> **Note**
>
> リソースを削除する際に、Evidently サービスでそのロールが使用されている場合、削除が失敗することがあります。失敗した場合は、数分待ってから操作を再試行してください。

AWSServiceRoleForCloudWatchEvidently で使用されている Evidently リソースを削除するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで [Application monitoring] (アプリケーションのモニターリング)、[Evidently] を選択します。
3. プロジェクトのリストで、シックライアントを使用したプロジェクトの横にあるチェックボックスを選択します。
4. [Project actions] (プロジェクトアクション)、[Delete project] (プロジェクトの削除) を選択します。

IAM を使用してサービスリンクロールを手動で削除するには

IAM コンソール、AWS CLI、または AWS API を使用して、AWSServiceRoleForCloudWatchEvidently サービスリンクロールを削除します。詳細については、IAM ユーザーガイドの「サービスにリンクされたロールの削除」を参照してください。

Evidently のサービスにリンクされたロールをサポートするリージョン

Evidently は、サービスが利用可能なすべてのリージョンで、サービスにリンクされたロールの使用をサポートします。詳細については、「AWS リージョンとエンドポイント」を参照してください。

CloudWatch Evidently クォータ

CloudWatch Evidently には、以下のクォータがあります。

リソース	デフォルトのクォータ
プロジェクト	リージョンごと、アカウントごとに 50 クォータは、引き上げをリクエストすることができます。
セグメント	リージョンごと、アカウントごとに 500 クォータは、引き上げをリクエストすることができます。
プロジェクトごとのクォータ	• 機能は合計 100 • 起動は合計 500 • 実行する起動は 50 • 実験は 合計 500 • 実行する実験は 50 これらのクォータはすべて、リクエストによって引き上げることができます。
API クォータ (リージョンごとのすべてのクォータ)	• PutProjectEvents: 米国東部 (バージニア北部)、米国西部 (オレゴン)、欧州 (アイルランド) では、1000 TPS (1 秒あたりのトランザクション件数) です。その他すべてのリージョンでは 200 TPS です。 • EvaluateFeature: 米国東部 (バージニア北部)、米国西部 (オレゴン)、欧州 (アイルランド) では、1000 TPS です。その他すべてのリージョンでは 200 TPS です。 • BatchEvaluateFeature: 50 TPS

リソース	デフォルトのクォータ
	• Create、Read、Update、Delete (CRUD) の API: CRUD API 全体で 10 TPS これらのクォータはすべて、リクエストによって引き上げることができます。

チュートリアル: Evidently のサンプルアプリケーションを使用した A/B テスト

このセクションでは、A/B テストに Amazon CloudWatch Evidently を使用するチュートリアルを提供します。このチュートリアルでは、シンプルな React アプリケーションである Evidently のサンプルアプリケーションを使用します。サンプルアプリケーションで、`showDiscount` 機能を表示するかどうかを設定します。ユーザーがこの機能を表示すると、ショッピングサイトで表示される価格が 20% の割引価格で表示されます。

このチュートリアルでは、一部のユーザーに割引を表示して他のユーザーには表示しなくすることに加え、両方のバリエーションからページのロード時間のメトリクスを収集するように Evidently を設定します。

> **Warning**
>
> このシナリオでは、プログラムによるアクセスと長期的な認証情報を持つ IAM ユーザーが必要です。これはセキュリティ上のリスクをもたらします。このリスクを軽減するために、これらのユーザーにはタスクの実行に必要な権限のみを付与し、不要になったユーザーを削除することをお勧めします。アクセスキーは、必要に応じて更新できます。詳細については、「IAM ユーザーガイド」の「アクセスキーの更新」を参照してください。

ステップ 1: サンプル アプリケーションのダウンロード

まず、Evidently のサンプルアプリケーションをダウンロードします。

サンプルアプリケーションをダウンロードするには

1. 次の Amazon S3 バケットから、サンプルアプリケーションをダウンロードします。

```
https://evidently-sample-application.s3.us-west-2.amazonaws.com/evidently-sample-shopping-app.zip
```

2. パッケージを解凍します。

ステップ 2: Evidently エンドポイントの追加と認証情報の設定

次に、以下の例のように、サンプルアプリケーションのパッケージの src ディレクトリにある config.js ファイルに、Evidently の リージョンとエンドポイントを追加します。

```
evidently: {
    REGION: "us-west-2",
    ENDPOINT: "https://evidently.us-west-2.amazonaws.com (https://evidently.us-west-2.amazonaws.com/)",
},
```

また、アプリケーションに CloudWatch を呼び出すためのアクセス許可があることを確認する必要があります。

サンプルアプリケーションに Evidently を呼び出すためのアクセス許可を付与するには

1. AWS アカウントにフェデレートします。
2. IAM ユーザーを作成し、AmazonCloudWatchEvidentlyFullAccess ポリシーをそのユーザーにアタッチします。
3. 次のステップで必要になるため、IAM ユーザーのアクセスキー ID とシークレットアクセスキーを書きとめておきます。
4. 次の例のように、このセクションの前半で変更したものと同じ config.js ファイルに、アクセスキー ID とシークレットアクセスキーの値を入力します。

```
credential: {
   accessKeyId: "Access key ID",
   secretAccessKey: "Secret key"
}
```

> **Important**
>
> このステップを使用すると、サンプルアプリケーションを可能な限り簡単に試すことができます。実際の本番アプリケーションに IAM ユーザーの認証情報を組み込むことは

お勧めしません。代わりに、Amazon Cognito を認証に使用することをお勧めします。詳細については、「Amazon Cognito のウェブアプリケーションとモバイルアプリケーションとの統合」を参照してください。

ステップ 3: 機能評価用のコードをセットアップする

CloudWatch Evidently を使用して機能を評価する場合は、[EvaluateFeature] オペレーションを使用して、ユーザーセッションごとに機能のバリエーションをランダムに選択する必要があります。このオペレーションは、実験で指定した割合に従って、機能の各バリエーションにユーザーセッションを割り当てます。

ブックストアデモアプリケーションの機能評価コードをセットアップするには

1. サンプルアプリケーションが Evidently を呼び出せるようにするために、`src/App.jsx` ファイルにクライアントビルダーを追加します。

```
import Evidently from 'aws-sdk/clients/evidently';
import config from './config';

const defaultClientBuilder = (
    endpoint,
    region,
) => {
    const credentials = {
        accessKeyId: config.credential.accessKeyId,
        secretAccessKey: config.credential.secretAccessKey
    }
    return new Evidently({
        endpoint,
        region,
        credentials,
    });
};
```

2. `const App` コードセクションに以下を追加して、クライアントを起動します。

```
  if (client == null) {
      client = defaultClientBuilder(
        config.evidently.ENDPOINT,
        config.evidently.REGION,
```

```
      );
```

3. 次のコードを追加して、evaluateFeatureRequest を作成します。このコードで、チュートリアルの後半で推奨するプロジェクト名と機能名を事前に入力しておきます。Evidently コンソールでプロジェクト名と機能名を指定すれば、独自のプロジェクト名と機能名に置き換えることができます。

```
const evaluateFeatureRequest = {
              entityId: id,
              // Input Your feature name
              feature: 'showDiscount',
              // Input Your project name'
              project: 'EvidentlySampleApp',
            };
```

4. 機能評価のために Evidently を呼び出すコードを追加します。リクエストが送信されると、Evidently は showDiscount 機能を表示するかどうかを指定するためにユーザーセッションをランダムに割り当てます。

```
client.evaluateFeature(evaluateFeatureRequest).promise().then(res => {
    if(res.value?.boolValue !== undefined) {
        setShowDiscount(res.value.boolValue);
    }
    getPageLoadTime()
 })
```

ステップ 4: 実験メトリクスのコードのセットアップ

カスタムメトリクスの場合は、Evidently のPutProjectEvents API を使用してメトリクスの結果を Evidently に送信します。次の例では、カスタムメトリクスを設定し、実験データを Evidently に送信する方法を示しています。

以下の関数を追加してページのロード時間を計算し、PutProjectEvents を使用してメトリクスの値を Evidently に送信します。次の機能を Home.tsx に追加し、EvaluateFeature API でこの関数を呼び出します。

```
const getPageLoadTime = () => {
    const timeSpent = (new Date().getTime() - startTime.getTime()) * 1.000001;
    const pageLoadTimeData = `{
      "details": {
```

```
        "pageLoadTime": ${timeSpent}
      },
      "UserDetails": { "userId": "${id}", "sessionId": "${id}"}
    }`;
    const putProjectEventsRequest = {
      project: 'EvidentlySampleApp',
      events: [
        {
          timestamp: new Date(),
          type: 'aws.evidently.custom',
          data: JSON.parse(pageLoadTimeData)
        },
      ],
    };
    client.putProjectEvents(putProjectEventsRequest).promise();
  }
```

ダウンロード後、編集した App.js ファイルは次のようになります。

```
import React, { useEffect, useState } from "react";
import { BrowserRouter as Router, Switch } from "react-router-dom";
import AuthProvider from "contexts/auth";
import CommonProvider from "contexts/common";
import ProductsProvider from "contexts/products";
import CartProvider from "contexts/cart";
import CheckoutProvider from "contexts/checkout";
import RouteWrapper from "layouts/RouteWrapper";
import AuthLayout from "layouts/AuthLayout";
import CommonLayout from "layouts/CommonLayout";
import AuthPage from "pages/auth";
import HomePage from "pages/home";
import CheckoutPage from "pages/checkout";
import "assets/scss/style.scss";
import { Spinner } from 'react-bootstrap';

import Evidently from 'aws-sdk/clients/evidently';
import config from './config';

const defaultClientBuilder = (
  endpoint,
  region,
) => {
  const credentials = {
```

```
    accessKeyId: config.credential.accessKeyId,
    secretAccessKey: config.credential.secretAccessKey
  }
  return new Evidently({
    endpoint,
    region,
    credentials,
  });
};

const App = () => {
  const [isLoading, setIsLoading] = useState(true);
  const [startTime, setStartTime] = useState(new Date());
  const [showDiscount, setShowDiscount] = useState(false);
  let client = null;
  let id = null;

  useEffect(() => {
    id = new Date().getTime().toString();
    setStartTime(new Date());
    if (client == null) {
      client = defaultClientBuilder(
        config.evidently.ENDPOINT,
        config.evidently.REGION,
      );
    }
    const evaluateFeatureRequest = {
      entityId: id,
      // Input Your feature name
      feature: 'showDiscount',
      // Input Your project name'
      project: 'EvidentlySampleApp',
    };

    // Launch
    client.evaluateFeature(evaluateFeatureRequest).promise().then(res => {
      if(res.value?.boolValue !== undefined) {
        setShowDiscount(res.value.boolValue);
      }
    });

    // Experiment
    client.evaluateFeature(evaluateFeatureRequest).promise().then(res => {
      if(res.value?.boolValue !== undefined) {
```

```
        setShowDiscount(res.value.boolValue);
      }
      getPageLoadTime()
    })

    setIsLoading(false);
  },[]);

  const getPageLoadTime = () => {
    const timeSpent = (new Date().getTime() - startTime.getTime()) * 1.000001;
    const pageLoadTimeData = `{
      "details": {
        "pageLoadTime": ${timeSpent}
      },
      "UserDetails": { "userId": "${id}", "sessionId": "${id}"}
    }`;
    const putProjectEventsRequest = {
      project: 'EvidentlySampleApp',
      events: [
        {
          timestamp: new Date(),
          type: 'aws.evidently.custom',
          data: JSON.parse(pageLoadTimeData)
        },
      ],
    };
    client.putProjectEvents(putProjectEventsRequest).promise();
  }
  return (
    !isLoading? (
    <AuthProvider>
      <CommonProvider>
        <ProductsProvider>
          <CartProvider>
            <CheckoutProvider>
              <Router>
                <Switch>
                  <RouteWrapper
                    path="/"
                    exact
                    component={() => <HomePage showDiscount={showDiscount}/>}
                    layout={CommonLayout}
                  />
                  <RouteWrapper
```

```
                    path="/checkout"
                    component={CheckoutPage}
                    layout={CommonLayout}
                  />
                  <RouteWrapper
                    path="/auth"
                    component={AuthPage}
                    layout={AuthLayout}
                  />
                </Switch>
              </Router>
            </CheckoutProvider>
          </CartProvider>
        </ProductsProvider>
      </CommonProvider>
    </AuthProvider> ) : (
      <Spinner animation="border" />
    )
  );
};

export default App;
```

ユーザーがサンプルアプリケーションにアクセスするたびに、カスタムメトリクスが Evidently に送信され分析されます。Evidently は各メトリクスを分析し、Evidently ダッシュボードにリアルタイムで結果を表示します。次の例で、メトリクスペイロードを示します。

```
[ {"timestamp": 1637368646.468, "type": "aws.evidently.custom", "data": "{\"details
\":{\"pageLoadTime\":2058.002058},\"userDetails\":{\"userId\":\"1637368644430\",
\"sessionId\":\"1637368644430\"}}" } ]
```

ステップ 5: プロジェクト、機能、実験の作成

次に、CloudWatch Evidently コンソールでプロジェクト、機能、および実験を作成します。

このチュートリアルのプロジェクト、機能、および実験を作成するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで、[Application Signals]、[Evidently] の順に選択します。
3. [Create project] (プロジェクトを作成) を選択し、フィールドに入力します。サンプルが正しく動作するためには、プロジェクト名に **EvidentlySampleApp** を使用する必要がありま

す。[Evaluation event storage] (評価イベントストレージ) で、[Don't store Evaluation events] (評価イベントを保存しない) を選択します。

フィールドに入力したら、[Create profile] (プロファイルを作成) を選択します。

詳細については、「 の新規プロジェクトの作成」を参照してください。

4. プロジェクトを作成したら、そのプロジェクトに機能を作成します。機能の名前は「**showDiscount**」にします。この機能では、**Boolean** 型のバリエーションを 2 つ作成します。最初のバリエーションの名前を「**disable**」、値を「**False**」とし、第 2 のバリエーションの名前を「**enable**」、値を「**True**」とします。

   機能の作成の詳細については、「プロジェクトに機能を追加する」を参照してください。

5. 機能の作成が完了したら、プロジェクト内の実験を作成します。実験の名前は「**pageLoadTime**」にします。

   この実験では、テスト対象ページのページロード時間を測定する `pageLoadTime` というカスタムメトリクスを使用します。実験のカスタムメトリクスは、Amazon EventBridge を使用して作成されます。EventBridge の詳細については、「Amazon EventBridge とは」を参照してください。

   そのカスタムメトリクスを作成するには、実験の作成時に次の操作を行います。

   - [Metrics] (メトリクス) の下で、 [Metric source] (メトリクスソース) として [Custom metrics] (カスタムメトリクス) を選択します。
   - [Metric name] (メトリクス名) に、「**pageLoadTime**」を入力します。
   - [Goal] (目標) には [Decrease] (減少) を選択します。これは、このメトリクスの値が小さいほど機能のバリエーションが最適であることを示します。
   - [Metric rule] (メトリクスルール) では、次のように入力します。
     - エンティティ IDには、**UserDetails.userId**と入力します。
     - [Value key] (値キー) には「**details.pageLoadTime**」と入力します。
     - [Units] (単位) には、「**ms**」と入力します。
   - [Add metric] (メトリクスを追加) を選択します。

   [Audiences] (対象者) には [100%] を選択し、すべてのユーザーが実験に参加するようにします。バリエーション間のトラフィック分割をそれぞれ 50% に設定します。

次に、[Create experiment] (実験を作成) を選択して、実験を作成します。作成しても、Evidently に開始するように指示するまで実験は開始されません。

ステップ 6: 実験を開始し、CloudWatch Evidently をテストする

最後のステップでは、実験を開始してサンプルアプリケーションを起動します。

チュートリアルの実験を開始するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで、[Application Signals]、[Evidently] の順に選択します。
3. [EvidentlySampleApp] プロジェクトを選択します。
4. [Experiments] (実験) タブを選択します。
5. [pageLoadTime] の横にあるボタンをクリックし、[Actions] (アクション)、[Start experiment] (実験を開始) を選択します。
6. 実験が終了する時間を選択します。
7. [Start experiment] (実験を開始) を選択します。

   実験がすぐに開始されます。

次に、以下のコマンドを使用して Evidently サンプルアプリケーションを起動します。

```
npm install -f && npm start
```

アプリケーションが起動すると、テストされる 2 つの機能のバリエーションのうちの 1 つに割り当てられます。1 つのバリエーションでは「20% 割引」と表示されますが、もう一方のバリエーションでは表示されていません。ページの更新を繰り返して、さまざまなバリエーションを表示します。

> **Note**
>
> Evidently にはスティッキーの評価があります。機能の評価は決定論的です。つまり、同じ `entityId` と機能では、必ず同じバリエーションがユーザーに割り当てられます。時間変動の割り当てが変更されるのは、エンティティがオーバーライドに追加されるか、実験トラフィックがダイヤルアップされる場合のみです。

ただし、サンプルアプリケーションのチュートリアルでの使用を簡単にするために、ページを更新するたびに Evidently によってサンプルアプリケーションの機能の評価が再割り当てされているため、オーバーライドを追加しなくても両方のバリエーションを体験できます。

トラブルシューティング

npm バージョン 6.14.14 を使用することをお勧めします。サンプルアプリケーションの構築または起動に関するエラーが表示されており、別のバージョンの npm を使用している場合は、次の手順を実行します。

npm バージョン 6.14.14 をインストールするには

1. ブラウザを使用して、https://nodejs.org/download/release/v14.17.5/ に接続します。
2. node-v14.17.5.pkg をダウンロードし、pkg ファイルを開いて npm をインストールします。

   `webpack not found` エラーが表示された場合は、`evidently-sample-shopping-app` フォルダを開き、以下のことを試します。

   a. `package-lock.json` を削除する
   b. `yarn-lock.json` を削除する
   c. `node_modules` を削除する
   d. `package.json` から webpack の依存関係を削除する
   e. 下記を実行します。

   ```
   npm install -f && npm
   ```

CloudWatch RUM を使用する

CloudWatch RUM を使用すると、現実的なユーザーモニターリングを実行できます。ウェブアプリケーションのパフォーマンスに関するクライアント側のデータを、ほぼリアルタイムで実際のユーザーセッションから収集し、それを表示できます。ページの読み込み時間、クライアント側のエラー、およびユーザーの活動などのデータを、可視化および分析することが可能です。この表示機能では、すべてのデータを集約的に見ることができ、同時に、顧客が使用しているブラウザやデバイスに関する詳細も確認できます。

収集されたデータは、クライアント側で発生するパフォーマンスの問題をすばやく特定し、デバッグを行う際に利用できます。CloudWatch RUM により、アプリケーションのパフォーマンス上の異常を可視化できます。さらに、エラーメッセージ、スタックトレース、ユーザーセッションなど、関連性のあるデバッグデータを見つけ出すこともできます。また RUM を使用すると、ユーザー数、地域的な位置、使用されているブラウザなど、どの範囲でエンドユーザーが影響を受けているかを把握することも可能です。

CloudWatch RUM により収集されたエンドユーザーデータは、30 日間保持された後で自動的に削除されます。RUM イベントをより長く保持する場合は、アプリケーションモニターで、イベントのコピーをアカウントの CloudWatch Logs に送信させることもできます。その上で、送信先ロググループの保持期間を調整します。

RUM を使用するには、アプリケーションモニターを作成し、情報をいくつか設定します。RUM では、アプリケーションに貼り付けるための JavaScript スニペットが自動的に生成されます。このスニペットは RUM のウェブクライアントコードを読み込みます。この RUM ウェブクライアントは、事前構築済みダッシュボードに表示されるアプリケーションのユーザーセッションから、その割合のデータをキャプチャします。ユーザーセッションのどの割合からデータを収集するか、指定することができます。

CloudWatch RUM は、アプリケーションサービス、クライアント、Synthetics Canary、サービスの依存関係を検出してモニタリングできる Application Signals と統合されています。Application Signals を使用すると、サービスのリストやビジュアルマップを確認したり、サービスレベル目標 (SLO) に基づくヘルスメトリクスを表示したり、ドリルダウンして相関関係のある X-Ray トレースを確認したりして、より詳細なトラブルシューティングを行うことができます。Application Signals で RUM クライアントページのリクエストを確認するには、アプリモニタを作成するか、RUM ウェブクライアントを手動で構成して、X-Ray アクティブトレースを有効にします。RUM クライアントは、サービスに接続されている [サービスマップ]、および呼び出されるサービスの [サービス詳細] ページに表示されます。

RUM ウェブクライアントはオープンソースです。詳細については、「CloudWatch RUM web client」を参照してください。

パフォーマンスに関する考慮事項

このセクションでは、CloudWatch RUM 使用上のパフォーマンスに関する考慮事項について説明します。

- ロードパフォーマンスへの影響 – CloudWatch RUM ウェブクライアントは、JavaScript モジュールとしてウェブアプリケーションにインストールされます。または、コンテンツ配信ネットワーク

(CDN) から非同期でウェブアプリケーションにロードされます。アプリケーション自体のロードプロセスが、阻害されることはありません。CloudWatch RUM は、アプリケーショがロードに要する時間に対して、検出可能な影響を与えることがないように設計されています。

- 実行時の影響 – RUM ウェブクライアントは、RUM データを記録し、CloudWatch RUM サービスにディスパッチするための処理を実行します。これらのイベントは頻度が低く処理量も多くありません。CloudWatch RUM は、アプリケーションのパフォーマンスが、この処理により検出可能な影響を受けることがないように設計されています。
- ネットワークへの影響 – RUM ウェブクライアントは、CloudWatch RUM サービスに定期的にデータを送信します。データは、アプリケーションの実行中、およびブラウザがアプリケーションをアンロードする直前に、一定の間隔で送出されます。ブラウザがアプリケーションをアンロードする直前に送信されるデータは、ビーコンとして送信されます。このビーコンが、アプリケーションのアンロード時間に検出可能な影響を与えることはありません。

RUM の使用料金

CloudWatch RUM では、このサービスが受け取るすべての RUM イベントに対して料金が発生します。RUM ウェブクライアントを使用して収集された、すべてのデータ項目は、RUM イベントと見なされます。RUM イベントの例としては、ページビュー、JavaScript エラー、および HTTP エラーなどが挙げられます。各アプリケーションモニターで収集されるイベントの種類を選択することが可能です。パフォーマンステレメトリイベント、JavaScript エラー、HTTP エラー、および X-Ray トレースの収集に関するオプションを、それぞれ有効または無効にできます。これらのオプション選択の詳細については、「ステップ 2: アプリケーションモニターを作成する」および「CloudWatch RUM ウェブクライアントによって収集される情報」を参照してください。料金の詳細については、「Amazon CloudWatch の料金」を参照してください。

利用可能なリージョン

CloudWatch RUM は現在、下記のリージョンで利用可能です。

- 米国東部 (バージニア北部)
- 米国東部 (オハイオ)
- 米国西部 (北カリフォルニア)
- 米国西部 (オレゴン)
- アフリカ (ケープタウン)
- アジアパシフィック (ジャカルタ)
- アジアパシフィック (ムンバイ)

- アジアパシフィック (ハイデラバード)
- アジアパシフィック (メルボルン)
- アジアパシフィック (大阪)
- アジアパシフィック (ソウル)
- アジアパシフィック (シンガポール)
- アジアパシフィック (シドニー)
- アジアパシフィック (東京)
- カナダ (中部)
- 欧州 (フランクフルト)
- 欧州 (アイルランド)
- 欧州 (ロンドン)
- 欧州 (ミラノ)
- 欧州 (パリ)
- 欧州 (スペイン)
- 欧州 (ストックホルム)
- 欧州 (チューリッヒ)
- 中東 (バーレーン)
- 中東 (アラブ首長国連邦)
- 南米 (サンパウロ)

トピック

- CloudWatch RUM で収集できる CloudWatch メトリクス
- CloudWatch RUM によるデータ保護とデータプライバシー
- CloudWatch RUM ウェブクライアントによって収集される情報
- CloudWatch RUM を使用するアプリケーションを管理する
- CloudWatch RUM でのクォータ
- CloudWatch RUM のトラブルシューティング

CloudWatch RUM 使用のための IAM ポリシー

CloudWatch RUM を完全に管理するには、IAM ポリシー AmazonCloudWatchRUMFullAccess を持つ、IAM ユーザーまたはロールとしてサインインする必要があります。さらに、他のポリシーやアクセス許可が必要になる場合もあります。

- 承認用の新しい Amazon Cognito ID プールを作成するアプリケーションモニターを作成するには、管理者の IAM ロール、または IAM ポリシー AdministratorAccess が必要です。
- CloudWatch Logs にデータを送信するアプリケーションモニターを作成するには、以下のアクセス許可を持つ IAM ロールまたはポリシーにログオンする必要があります。

```
{
    "Effect": "Allow",
    "Action": [
        "logs:PutResourcePolicy"
    ],
    "Resource": [
        "*"
    ]
}
```

CloudWatch RUM データを表示する必要があるものの、CloudWatch RUM リソースの作成は行わない他のユーザーについては、AmazonCloudWatchRUMReadOnlyAccess ポリシーを使用できます。

CloudWatch RUM を使用するためにアプリケーションをセットアップする

ユーザーセッションから実際のパフォーマンスデータを CloudWatch RUM が収集できるように、アプリケーションのセットアップを行うには、以下のセクションの各ステップを実行します。

トピック

ステップ 1: AWS にデータを送信することを、アプリケーションに許可する

CloudWatch RUM を使用するアプリケーションには、認証が必要です。

この認証を設定するためには、以下の 3 つのオプションがあります。

- CloudWatch RUM に、アプリケーションの新しい Amazon Cognito ID プールを作成させます。この方法では、セットアップに必要な労力が最小限に抑えられます。これがデフォルトのオプションです。

  ID プールには、認証されていない ID が含まれています。これにより、CloudWatch RUM のウェブクライアントは、アプリケーションのユーザーを認証することなく CloudWatch RUM にデータを送信できます。

  Amazon Cognito ID プールには、アタッチ済みの IAM ロールも含まれています。Amazon Cognito 内の認証されていない ID により、ウェブクライアントは、CloudWatch RUM にデータを送信することが許可された IAM ロールを引き受けることができます。
- 既存の Amazon Cognito ID プールを使用する この場合、ID プールにアタッチされている IAM ロールを、同時に変更する必要があります。このオプションは、認証されていないユーザーをサポートするアイデンティティプールに使用します。ID プールは、同じリージョンからのみ使用できます。
- 先に設定を行ってある、既存の ID プロバイダからの認証を使用します。この場合は、ID プロバイダから認証情報を取得する必要があります。またこれらの認証情報は、アプリケーションから RUM ウェブクライアントに転送する必要があります。

  認証されたユーザーのみをサポートするアイデンティティプールには、このオプションを使用します。

以下のセクションで、これらのオプションについてさらに詳しく説明します。

CloudWatch RUM が、新しい Amazon Cognito ID プールを作成します。

これが、設定のための最も簡単なオプションで、これを選択した場合は、それ以上のセットアップ作業は必要ありません。このオプションを使用するには、管理用のアクセス許可が必要です。詳細については、「CloudWatch RUM 使用のための IAM ポリシー」を参照してください。

このオプションを使用すると、CloudWatch RUM は以下のリソースを作成します。

- 新しい Amazon Cognito ID プール
- 認証されていない Amazon Cognito ID これにより RUM のウェブクライアントは、アプリケーションのユーザーを認証することなく、IAM ロールを引き受けることが可能になります。
- RUM ウェブクライアントが引き受ける IAM ロール。このロールにアタッチされた IAM ポリシーにより、アプリケーションモニターのリソースを使用しながら PutRumEvents APIを実行することが許可されます。つまり、RUM ウェブクライアントから RUM にデータを送信できるようにします。

RUM ウェブクライアントは、Amazon Cognito アイデンティティを使用して、AWS の認証情報を取得します。取得された AWS 認証情報は、対象の IAM ロールに関連付けられます。この IAM ロールでは、AppMonitor リソースに対し PutRumEvents を使用することが許可されています。

Amazon Cognito は、アプリケーションが CloudWatch RUM にデータを送信できるようにするための、必要なセキュリティトークンを送信します。CloudWatch RUM が生成する JavaScript コードスニペットには、以下のような、認証を有効にするための行が含まれています。

```
   {
    identityPoolId: [identity pool id],  // e.g., 'us-west-2:EXAMPLE4a-66f6-4114-902a-
EXAMPLEbad7'
   }
);
```

既存の Amazon Cognito ID プールを使用する

既存の Amazon Cognito ID プールを使用する場合は、アプリケーションを CloudWatch RUM に追加する際に、その ID プールを指定します。このプールは、認証されていない ID に対するアクセスの有効化が、サポートされている必要があります。ID プールは、同じリージョンからのみ使用できます。

また、この ID プールに関連付けられている IAM ロールにアタッチされている IAM ポリシーに対し、次のアクセス許可を追加する必要があります。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "rum:PutRumEvents",
            "Resource": "arn:aws:rum:[region]:[accountid]:appmonitor/[app monitor
 name]"
        }
    ]
}
```

これを設定することで Amazon Cognito は、アプリケーションが CloudWatch RUM にアクセスできるようにするための、必要なセキュリティトークンを送信するようになります。

サードパーティのプロバイダー

サードパーティーのプロバイダーからのプライベート認証の使用を選択する場合は、ID プロバイダーから認証情報を取得し、AWS に転送する必要があります。これを行う最良の方法の 1 つは、セキュリティトークンベンダーを利用することです。これには、AWS Security Token Service を使用する Amazon Cognito を含め、任意のセキュリティトークンベンダーが使用できます。AWS STS の詳細については、「Welcome to the AWS Security Token Service API Reference」参照してください。

このシナリオで Amazon Cognito をトークンベンダーとして使用する場合は、認証プロバイダーと連携するように Amazon Cognito を設定します。詳細については、「Amazon Cognito ID プール (フェデレーティッド ID) の使用開始方法」を参照してください。

Amazon Cognito で ID プロバイダーとの連携を設定し終えたら、以下の操作も実行する必要があります。

- 以下のアクセス許可を持つ IAM ロールを作成します。アプリケーションは、AWS へのアクセスにこのロールを使用します。

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

```
            "Effect": "Allow",
            "Action": "rum:PutRumEvents",
            "Resource": "arn:aws:rum:[region]:[accountID]:appmonitor/[app monitor
 name]"
        }
    ]
}
```

- アプリケーションがプロバイダーからの認証情報を CloudWatch RUM に渡すようにするため。以下を追加します。アプリケーションに行を挿入します。これは、ユーザーがアプリケーションにサインインし、AWS へのアクセスに使用する資格情報がアプリケーションで受け取られた後に実行されます。

```
cwr('setAwsCredentials', {/* Credentials or CredentialProvider */});
```

AWS JavaScript SDK での認証情報プロバイダーの詳細については、SDK for JavaScript v3 デベロッパーガイドの「ウェブブラウザでの認証情報の設定」、SDK for JavaScript v2 デベロッパーガイドの「ウェブブラウザでの認証情報の設定」、および @aws-sdk/credential-providers を参照してください。

また、CloudWatch RUM ウェブクライアント向け SDK を使用して、ウェブクライアントの認証方法を設定することもできます。ウェブクライアント SDK の詳細については、「CloudWatch RUM web client SDK」を参照してください。

ステップ 2: アプリケーションモニターを作成する

アプリケーションで CloudWatch RUM の使用を開始するには、アプリケーションモニター を作成します。アプリケーションモニターが作成されると、アプリケーションに貼り付けるための JavaScript スニペットが、RUM により生成されます。このスニペットは RUM のウェブクライアントコードを読み込みます。RUM のウェブクライアントは、アプリケーションのユーザーセッションの割合に関するデータをキャプチャし、それを RUM に送信します。

アプリケーションモニターを作成するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで、[Application Signals]、[RUM] の順に選択します。
3. [Add app monitor] (アプリケーションモニターを追加) をクリックします。
4. アプリケーションの情報と設定を入力します。

- [App monitor name] (アプリケーションモニター名) に、CloudWatch RUM コンソール内でこのモニターを識別する際に使用する名前を入力します。
- [Application domain] (アプリケーションドメイン) で、アプリケーションが管理権限を持つ最上位レベルのドメイン名を入力します。この情報は URL ドメイン形式であることが必要です。

  [Include sub domains] (サブドメインを含める) を選択し、トップレベルドメインの下に存在するすべてのサブドメインから、アプリケーションモニターがデータを収集するようにします。

5. [Configure RUM data collection] (RUM データ収集の設定) で、以下の各情報をアプリケーションモニターに収集させるかどうかを指定します。

   - パフォーマンステレメトリ – ページならびにリソースのロード時間に関する情報を収集します。
   - JavaScript エラー – アプリケーションによって発生した未処理の JavaScript エラーに関する情報を収集します。
   - HTTP エラー – アプリケーションによってスローされた HTTP エラーに関する情報を収集します。

   これらのオプションを選択することで、アプリケーションに関する詳細情報を得ることができますが、同時により多くの CloudWatch RUM イベントが生成されるため、発生する料金も増加します。

   これらのいずれかを選択しない場合でも、アプリケーションモニターはセッション開始イベントとページ ID を収集します。これにより、アプリケーションを使用しているユーザーの数を詳細情報 (オペレーティングシステムの種類とバージョン、ブラウザの種類とバージョン、デバイスの種類、および場所など) ごとに確認できます。

6. サンプリングされたユーザーセッションからの、ユーザー ID とセッション ID の収集を可能にする場合には、[Check this option to allow the CloudWatch RUM Web Client to set cookies] (CloudWatch RUM ウェブクライアントでクッキーを設定する場合このオプションをオンにします) を選択します。ユーザー ID は RUM によってランダムに生成されます。詳細については、「CloudWatch RUM ウェブクライアント Cookie (または類似の技術)」を参照してください。
7. [Session samples] (セッションサンプル) で、RUM データの収集に使用されるユーザーセッションの割合を入力します。デフォルトでこの値は 100% に設定されています。この数を減らすと、取得されるデータは少なくなりますが料金が削減されます。RUM での料金の詳細については、「RUM pricing」を参照してください。

8. CloudWatch RUM 用に収集したエンドユーザーデータは 30 日間保持され、その後削除されます。RUM イベントのコピーを CloudWatch Logs に保存し、そのコピーの保持期間を設定する場合は、[Data storage] (データストレージ) で、[Check this option to store your application telemetry data in your CloudWatch Logs account] (テレメトリデータを CloudWatch Logs アカウントに保存するにはこのオプションをオンにします) を選択します。デフォルトでは、CloudWatch Logs のロググループは 30 日間データを保持します。ログの保持期間は、CloudWatch Logs コンソールで管理できます。

9. [Authorization] (認証) で、(新規または既存の) Amazon Cognito ID プールを使用するか、異なる ID プロバイダーを使用するかを指定します。新しい ID プールの作成は、他のセットアップ手順を必要としない最も簡単なオプションです。詳細については、「ステップ 1: AWS にデータを送信することを、アプリケーションに許可する」を参照してください。

   Amazon Cognito ID プールの新規作成には、管理者権限が必要です。詳細については、「CloudWatch RUM 使用のための IAM ポリシー」を参照してください。

10. (オプション) デフォルトでは、アプリケーションに RUM コードスニペットを追加すると、ウェブクライアントにより、そのアプリケーションのすべてのページの HTML コード内にモニターリング用の JavaScript タグが挿入されます。これを変更するには、[Configure pages] (ページの設定) をクリックした後に、[Include only these pages] (これらのページのみを含める)、または[Exclude these pages] (これらのページを除外する) のどちらかを選択します。その上で、含める、もしくは除外するページを指定します。含めるか除外するページを指定するには、その完全な URL を入力します。追加のページを指定するには、[Add URL] (URLを追加) をクリックします。

11. アプリケーションモニターによってサンプリングされたユーザーセッションで AWS X-Ray トレースを有効化するには、[Active tracing] (アクティブなトレース) をクリックし、[Trace my service with AWS X-Ray] を選択します。

    これを選択すると、ユーザーセッション中に発行されアプリモニターによってサンプリングされた、`XMLHttpRequest` および `fetch` リクエストがトレースされるようになります。その後、これらのユーザーセッションについてのトレースとセグメントを、RUM ダッシュボード、X-Ray トレースマップページ、X-Ray トレースの詳細ページ上に表示できるようになります。これらのユーザーセッションは、Application Signals でアプリケーションを有効にすると、そこにクライアントページとしても表示されます。

    CloudWatch RUM ウェブクライアントの設定をさらに変更することで、HTTP リクエストに X-Ray トレースヘッダーを追加し、AWS のマネージド型サービスにおけるユーザーセッションを、下流方向にエンドツーエンドでトレースできるようになります。詳細については、「X-Ray でのエンドツーエンドのトレースを有効にする」を参照してください。

12. (オプション) アプリケーションモニターにタグを追加する場合は、[Tags] (タグ)、[Add new tag] (新しいタグを追加) の順にクリックします。

    次に、[Key] (キー) でタグの名前を入力します。[値] では、任意でタグに値を追加できます。

    (オプション) 別のタグを追加するには、[Add new tag] (新しいタグを追加) を再度選択します。

    詳細については、「AWS リソースのタグ付け」を参照してください。
13. [Add app monitor] (アプリケーションモニターを追加) をクリックします。
14. [Sample code] (サンプルコード) セクションで、アプリケーションに追加するために使用するコードスニペットをコピーできます。[JavaScript] または [TypeScript] を選択し、NPM を使用して、CloudWatch RUM ウェブクライアントを JavaScript モジュールとしてインストールすることをお勧めします。

    または、[HTML] を選択し、コンテンツ配信ネットワーク (CDN) を使用して、CloudWatch RUM ウェブクライアントをインストールすることもできます。CDN を使用するデメリットは、Web クライアントが広告ブロッカーによって頻繁にブロックされることです。
15. [Copy] (コピー) または [Download] (ダウンロード)、続いて [Done] (完了) をクリックします。

(オプション) ステップ 3: コードスニペットを手動で変更して CloudWatch RUM ウェブクライアントを構成する

アプリケーションに挿入する前のコードスニペットを変更して、いくつかのオプションを有効または無効にすることができます。詳細については、「CloudWatch RUM web client documentation」を参照してください。

この場合には、次のセクションで説明するように、配慮しておくべき設定オプションが 3 つ存在します。

個人情報を含む可能性のあるリソース URL の収集を防止する

デフォルトでは、CloudWatch RUM ウェブクライアントは、アプリケーションによってダウンロードされたリソースの URL を記録するように設定されています。これらのリソースには、HTML ファイル、画像、CSS ファイル、JavaScript ファイルなどが含まれます。アプリケーションによって、URL には個人を特定できる情報 (PII) が含まれている可能性があります。

アプリケーションがこのような構成の場合には、アプリケーションに挿入する前のコードスニペット内で `recordResourceUrl: false` を設定し、リソース URL のコレクションを無効にしておくことを強くお勧めします。

ページビューを手動で記録する

ウェブクライアントのデフォルト設定では、ページが最初にロードされた時点、ならびにブラウザの履歴 API が呼び出され時点でページビューが記録されます。デフォルトのページ ID は `window.location.pathname` です。ただし、場合によってはこの動作をオーバーライドしてアプリケーションを計測し、ページビューをプログラムで記録します。これにより、ページ ID とそれを記録するタイミングを管理できます。例えば、`/entity/123` または `/entity/456` などの変数識別子を持つ URI のウェブアプリケーションを考えてみましょう。デフォルトでは、CloudWatch RUM はパス名と一致する個別のページ ID を持つページビューイベントを URI ごとに生成しますが、代わりに同じページ ID でグループ化することもできます。これを実行するには、`disableAutoPageView` 設定を使用してウェブクライアントのページビュー自動化を無効にし、`recordPageView` コマンドを使用して目的のページ ID を設定します。詳細については、GitHub の「アプリケーション固有の設定」を参照してください。

埋め込みスクリプトの例:

```
cwr('recordPageView', { pageId: 'entityPageId' });
```

JavaScript モジュールの例:

```
awsRum.recordPageView({ pageId: 'entityPageId' });
```

X-Ray でのエンドツーエンドのトレースを有効にする

アプリケーションモニターを作成する際に、[Trace my service with AWS X-Ray] を選択します。これにより、アプリケーションモニターがサンプリングするユーザーセッション中に発行された `XMLHttpRequest` および `fetch` リクエストのトレースを有効化します。その後、これらの HTTP リクエストについてのトレースを、CloudWatch RUM ダッシュボード、X-Ray トレースマップページ、X-Ray トレースの詳細ページ上に表示できるようになります。

デフォルトでは、これらのクライアント側トレースは、サーバー側のダウンストリームトレースとは接続されません。クライアント側のトレースをサーバー側トレースに接続し、エンドツーエンドのトレースを有効にするには、ウェブクライアントの `addXRayTraceIdHeader` オプションで `true` を設定します。これにより、CloudWatch RUM ウェブクライアントは、HTTP リクエストに X-Ray トレースヘッダーを追加します。

次に、クライアント側のトレースを追加する場合のコードブロック例を示します。可読性のために、このサンプルでは一部の構成オプションが省略されています。

```
<script>
    (function(n,i,v,r,s,c,u,x,z){...})(
        'cwr',
        '00000000-0000-0000-0000-000000000000',
        '1.0.0',
        'us-west-2',
        'https://client.rum.us-east-1.amazonaws.com/1.0.2/cwr.js',
        {
            enableXRay: true,
            telemetries: [
                'errors',
                'performance',
                [ 'http', { addXRayTraceIdHeader: true } ]
            ]
        }
    );
</script>
```

> **Warning**
>
> HTTP リクエストに X-Ray トレースヘッダーを追加するための設定を、CloudWatch RUM のウェブクライアントで行うことで、クロスオリジンリソース共有 (CORS) の失敗を引き起こしたり、Sigv4 で署名されているリクエスト署名が無効化されたりする場合があります。詳細については、「CloudWatch RUM web client documentation」を参照してください。本番環境でクライアント側の X-Ray トレースヘッダーの追加を行う前に、アプリケーションのテストを実施することを強くお勧めします。

詳細については、「CloudWatch RUM web client documentation」を参照してください。

ステップ 4: コードスニペットをアプリケーションに挿入する

次に、前出のセクションで作成したコードスニペットをアプリケーションに挿入します。

> **Warning**
>
> コードスニペットによってダウンロードおよび構成されたウェブクライアントでは、Cookie (もしくは類似の技術) を利用して、エンドユーザーからのデータを収集できます。コードスニペットを挿入する前に、「コンソールでのメタデータ属性によるフィルタリング」を参照してください。

生成されたコードスニペットをまだ用意していない場合は、既に生成してあるコードスニペットを見つけるにはどうすればよいですか? の手順に従ってスニペット入手できます。

CloudWatch RUM のコードスニペットをアプリケーションに挿入するには

1. 前のセクションでコピーまたはダウンロードしたコードスニペットをアプリケーションの <head> 要素に挿入します。スニペットは、<body> 要素または他のすべての <script> タグより前に挿入してください。

   生成されるスクリプトの例を以下に示します。

```
<script>
(function (n, i, v, r, s, c, x, z) {
    x = window.AwsRumClient = {q: [], n: n, i: i, v: v, r: r, c: c};
    window[n] = function (c, p) {
        x.q.push({c: c, p: p});
    };
    z = document.createElement('script');
    z.async = true;
    z.src = s;
    document.head.insertBefore(z, document.getElementsByTagName('script')[0]);
})('cwr',
    '194a1c89-87d8-41a3-9d1b-5c5cd3dafbd0',
    '1.0.0',
    'us-east-2',
    'https://client.rum.us-east-1.amazonaws.com/1.0.2/cwr.js',
    {
        sessionSampleRate: 1,
        identityPoolId: "us-east-2:c90ef0ac-e3b8-4d1a-b313-7e73cfd21443",
        endpoint: "https://dataplane.rum.us-east-2.amazonaws.com",
        telemetries: ["performance", "errors", "http"],
        allowCookies: true,
        enableXRay: false
    });
</script>
```

2. 対象のウェブアプリケーションが複数のページを持つ場合は、データを収集した各 HTML ページごとに、ステップ 1 を繰り返す必要があります。

ステップ 5: ユーザーイベントを生成してアプリケーションモニターの設定内容をテストする

コードスニペットを挿入し、更新されたアプリケーションを実行したら、そのテストのためにユーザーイベントを手動で生成します。このテストでは、以下を実行することをお勧めします。このテストでは、標準の CloudWatch RUM 料金が発生します。

- ウェブアプリケーション内のページ間を移動する。
- 異なるブラウザとデバイスを使用しながら、複数のユーザーセッションを作成する。
- リクエストを発行する。
- JavaScript エラーを発生させる。

いくつかのイベントを生成したら、CloudWatch RUM ダッシュボードでそれらを確認します。詳細については、「CloudWatch RUM ダッシュボードの表示」を参照してください。

ユーザーセッションのデータがダッシュボードに表示されるまでには、最大で 15 分かかる場合があります。

アプリケーションでイベントを生成してから、15 分経過してもデータが表示されない場合は、CloudWatch RUM のトラブルシューティング を参照してください。

CloudWatch RUM ウェブクライアントの設定

アプリケーションでは、CloudWatch RUM によって生成されたコードスニペットの 1 つを使用して、CloudWatch RUM ウェブクライアントをインストールできます。生成されたスニペットは、NPM 経由の JavaScript モジュール、またはコンテンツ配信ネットワーク (CDN) の 2 つのインストール方法をサポートしています。ベストパフォーマンスを実現するため、NPM によるインストール方法を使用することをお勧めします。この方法を使用する詳細については、「Installing as a JavaScript Module」(JavaScript モジュールとしてインストールする) を参照してください。

CDN インストールオプションを使用すると、広告ブロッカーが、CloudWatch RUM によって提供されるデフォルトの CDN をブロックする場合があります。これにより、広告ブロッカーをインストールしているユーザーのアプリケーションモニターリングが無効になります。このため、デフォルトの CDN は、CloudWatch RUM を使用する初期オンボーディングにのみ使用することをお勧めします。この問題を軽減する方法の詳細については、「Instrument the application」(アプリケーションの計測) を参照してください。

コードスニペットは、HTML ファイルの <head> タグ内に配置されており、ウェブクライアントをダウンロードしてインストールします。さらにモニターリング対象のアプリケーション向けに、インストールしたウェブクライアントを設定します。このスニペットは、次のような自己実行が可能な関数です。この例では、読みやすさのためにスニペットの関数の本体は省略されています。

```
<script>
(function(n,i,v,r,s,c,u,x,z){...})(
'cwr',
'00000000-0000-0000-0000-000000000000',
'1.0.0',
'us-west-2',
'https://client.rum.us-east-1.amazonaws.com/1.0.2/cwr.js',
{ /* Configuration Options Here */ }
);
<script>
```

引数

このコードスニペットでは、以下の 6 つの引数を受け入れます。

- ウェブクライアントでコマンドを実行するための名前空間 (例: `'cwr'`)
- アプリケーションモニターの ID (例: `'00000000-0000-0000-0000-000000000000'`)
- アプリケーションのバージョン (例: `'1.0.0'`)
- アプリケーションモニターの AWS リージョン (例: `'us-west-2'`)
- ウェブクライアントの URL (例: `'https://client.rum.us-east-1.amazonaws.com/1.0.2/cwr.js'`)
- アプリケーションに固有の設定オプション 詳細については、以下の セクションを参照してください。

エラーの無視

CloudWatch RUM ウェブクライアントは、アプリケーションで発生するあらゆる種類のエラーをリッスンします。CloudWatch RUM ダッシュボードに表示したくない JavaScript エラーがアプリケーションから発行される場合、CloudWatch RUM ウェブクライアントを設定してこれらのエラーを除外し、関連するエラーイベントのみを CloudWatch RUM ダッシュボードに表示することができます。例えば、修正方法がすでに特定されている一部の JavaScript エラーが大量に発生して他のエラーを覆い隠しているような場合は、これらのエラーをダッシュボードに表示しないように選択でき

ます。また、サードパーティが所有するライブラリに帰属するために修正できないエラーも無視したい場合があります。

特定の JavaScript エラーを除外するためにウェブクライアントをインストルメントする方法の詳細については、ウェブクライアントの Github ドキュメントの「Errors」(エラー) の例を参照してください。

設定オプション

CloudWatch RUM ウェブクライアントで使用できる設定オプションの詳細については、CloudWatch RUM ウェブクライアントのドキュメントを参照してください。

地域化

このセクションでは、CloudWatch RUM をさまざまなリージョンのアプリケーションで使用する戦略について説明します。

ウェブアプリケーションが複数の AWS リージョンにデプロイされている

ウェブアプリケーションが複数の AWS リージョンにデプロイされている場合、以下の 3 つのオプションがあります。

- 1 つのリージョン、1 つのアカウントに 1 つのアプリモニタをデプロイして、すべてのリージョンにサービスを提供します。
- リージョンごとに一意のアカウントで、個別のアプリモニタをデプロイします。
- リージョンごとに、すべて 1 つのアカウントで、個別のアプリモニタをデプロイします。

1 つのアプリモニタを使用する利点は、すべてのデータが 1 つのビジュアライゼーションに一元化され、すべてのログが CloudWatch Logs の同じロググループに書き込まれることです。アプリモニタが 1 つしかないと、リクエストに若干のレイテンシーが発生し、単一障害点が発生します。

複数のアプリモニタを使用すると、単一障害点はなくなりますが、すべてのデータを 1 つのビジュアライゼーションにまとめることはできません。

アプリケーションがデプロイされている一部のリージョンで、CloudWatch RUM がリリースされていない

CloudWatch RUM は多くのリージョンでリリースされており、地理的にも広範囲に及んでいます。CloudWatch RUM が利用可能なリージョンにセットアップすることで、そのメリットを得るこ

とができます。接続先のリージョンにアプリモニタをセットアップしていれば、エンドユーザーはどこにいてもセッションを含めることができます。

ただし、CloudWatch RUM は、AWS GovCloud (米国東部)、AWS GovCloud (米国西部)、すべての中国リージョンではまだリリースされていません。これらのリージョンから CloudWatch RUM にデータを送信することはできません。

ページグループを使用する

ページグループを使用して、アプリケーション内の異なるページを相互に関連付けて、ページグループの集計分析を確認できるようにします。例えば、すべてのランディングページのロード時間の集計を確認したい場合があります。

ページをページグループに入れるには、CloudWatch RUM ウェブクライアントのページビューイベントに 1 つまたは複数のタグを追加します。次の例では、/home ページを en という名前のページグループと landing という名前のページグループに入れています。

埋め込みスクリプトの例

```
cwr('recordPageView', { pageId: '/home', pageTags: ['en', 'landing']});
```

JavaScript モジュールの例

```
awsRum.recordPageView({ pageId: '/home', pageTags: ['en', 'landing']});
```

> **Note**
>
> ページグループは、さまざまなページにわたる分析の集計を容易にすることを目的としています。アプリケーションに合わせて pageIds を定義し、操作する方法については、「(オプション) ステップ 3: コードスニペットを手動で変更して CloudWatch RUM ウェブクライアントを構成する」の「ページビューを手動で記録する」セクションを参照してください。

カスタムメタデータを指定する

CloudWatch RUM は、メタデータとして各イベントに追加データをアタッチします。イベントメタデータはキーと値のペアの形式の属性で構成されます。これらの属性を使用して、CloudWatch RUM コンソールでイベントを検索またはフィルタリングできます。デフォルトでは、CloudWatch RUM

はいくつかのメタデータを自動的に作成します。デフォルトメタデータの詳細については、「RUM イベントのメタデータ」を参照してください。

また、CloudWatch RUM ウェブクライアントを使用して、CloudWatch RUM イベントにカスタムメタデータを追加することもできます。カスタムメタデータには、セッション属性とページ属性を含めることができます。

カスタムメタデータを追加するには、CloudWatch RUM ウェブクライアントのバージョン 1.10.0 以降を使用する必要があります。

要件と構文

各イベントのメタデータには最大で 10 個のカスタム属性を含めることができます。カスタム属性の構文要件は次のとおりです。

- キー
  - 最大 128 文字
  - 英数字、コロン (:)、アンダースコア (_) を使用できます。
  - 先頭を `aws:` にすることはできません。
  - 次のセクションに記載されている予約キーワードのみで構成することはできません。これらのキーワードを長いキー名の一部として使用することはできます。
- [値]
  - 最大 256 文字
  - 文字列、数値、またはブール値である必要があります

予約キーワード

次の予約キーワードをキー名そのものとして使用することはできません。次のキーワードを `applicationVersion` のように長いキー名の一部として使用することはできます。

- `browserLanguage`
- `browserName`
- `browserVersion`
- `countryCode`
- `deviceType`
- `domain`

- interaction
- osName
- osVersion
- pageId
- pageTags
- pageTitle
- pageUrl
- parentPageId
- platformType
- referrerUrl
- subdivisionCode
- title
- url
- version

> **Note**
>
> CloudWatch RUM は、属性に無効なキーまたは値が含まれている場合、またはイベントあたりのカスタム属性の上限 10 個に既に達している場合、RUM イベントからカスタム属性を削除します。

セッション属性を追加する

カスタムセッション属性を設定すると、セッション内のすべてのイベントに追加されます。セッション属性は、CloudWatch RUM ウェブクライアントの初期化中または実行時に addSessionAttributes コマンドを使用して設定します。

例えば、アプリケーションのバージョンをセッション属性として追加できます。次に、CloudWatch RUM コンソールで、エラーをバージョン別にフィルタリングして、エラー率の増加がアプリケーションの特定のバージョンに関連しているかどうかを確認できます。

初期化時のセッション属性の追加、NPM の例

太字のコードセクションでセッション属性が追加されます。

```
import { AwsRum, AwsRumConfig } from 'aws-rum-web';

try {
  const config: AwsRumConfig = {
    allowCookies: true,
    endpoint: "https://dataplane.rum.us-west-2.amazonaws.com",
    guestRoleArn: "arn:aws:iam::000000000000:role/RUM-Monitor-us-
west-2-000000000000-00xx-Unauth",
    identityPoolId: "us-west-2:00000000-0000-0000-0000-000000000000",
    sessionSampleRate: 1,
    telemetries: ['errors', 'performance'],
    sessionAttributes: {
        applicationVersion: "1.3.8"
    }
  };

  const APPLICATION_ID: string = '00000000-0000-0000-0000-000000000000';
  const APPLICATION_VERSION: string = '1.0.0';
  const APPLICATION_REGION: string = 'us-west-2';

  const awsRum: AwsRum = new AwsRum(
    APPLICATION_ID,
    APPLICATION_VERSION,
    APPLICATION_REGION,
    config
  );
} catch (error) {
  // Ignore errors thrown during CloudWatch RUM web client initialization
}
```

実行時のセッション属性の追加、NPM の例

```
awsRum.addSessionAttributes({
    applicationVersion: "1.3.8"
})
```

初期化時のセッション属性の追加、埋め込みスクリプトの例

太字のコードセクションでセッション属性が追加されます。

```
<script>
    (function(n,i,v,r,s,c,u,x,z){...})(
        'cwr',
```

```
        '00000000-0000-0000-0000-000000000000',
        '1.0.0',
        'us-west-2',
        'https://client.rum.us-east-1.amazonaws.com/1.0.2/cwr.js',
        {
            sessionSampleRate:1,
            guestRoleArn:'arn:aws:iam::000000000000:role/RUM-Monitor-us-
west-2-000000000000-00xx-Unauth',
            identityPoolId:'us-west-2:00000000-0000-0000-0000-000000000000',
            endpoint:'https://dataplane.rum.us-west-2.amazonaws.com',
            telemetries:['errors','http','performance'],
            allowCookies:true,
            sessionAttributes: {
                applicationVersion: "1.3.8"
            }
        }
    );
</script>
```

実行時のセッション属性の追加、埋め込みスクリプトの例

```
<script>
    function addSessionAttribute() {
        cwr('addSessionAttributes', {
            applicationVersion: "1.3.8"
        })
    }

</script>
```

ページ属性を追加する

カスタムページ属性を設定すると、現在のページのすべてのイベントに追加されます。ページ属性は、CloudWatch RUM ウェブクライアントの初期化中または実行時に `recordPageView` コマンドを使用して設定します。

例えば、ページテンプレートをページ属性として追加できます。次に、CloudWatch RUM コンソールで、エラーをページテンプレート別にフィルタリングして、エラー率の増加がアプリケーションの特定のページテンプレートに関連しているかどうかを確認できます。

初期化時のページ属性の追加、NPM の例

太字のコードセクションでページ属性が追加されます。

```
const awsRum: AwsRum = new AwsRum(
    APPLICATION_ID,
    APPLICATION_VERSION,
    APPLICATION_REGION,
    { disableAutoPageView:  true // optional }
);
awsRum.recordPageView({
    pageId:'/home',
    pageAttributes: {
      template: 'artStudio'
    }
});
const credentialProvider = new CustomCredentialProvider();
if(awsCreds) awsRum.setAwsCredentials(credentialProvider);
```

実行時のページ属性の追加、NPM の例

```
awsRum.recordPageView({
    pageId: '/home',
    pageAttributes: {
        template: 'artStudio'
    }
});
```

初期化時のページ属性の追加、埋め込みスクリプトの例

太字のコードセクションでページ属性が追加されます。

```
<script>
    (function(n,i,v,r,s,c,u,x,z){...})(
        'cwr',
        '00000000-0000-0000-0000-000000000000',
        '1.0.0',
        'us-west-2',
        'https://client.rum.us-east-1.amazonaws.com/1.0.2/cwr.js',
        {
            disableAutoPageView: true //optional
        }
    );
    cwr('recordPageView', {
       pageId: '/home',
       pageAttributes: {
```

```
            template: 'artStudio'
        }
    });
    const awsCreds = localStorage.getItem('customAwsCreds');
    if(awsCreds) cwr('setAwsCredentials', awsCreds)
</script>
```

実行時のページ属性の追加、埋め込みスクリプトの例

```
<script>
    function recordPageView() {
        cwr('recordPageView', {
            pageId: '/home',
            pageAttributes: {
                template: 'artStudio'
            }
        });
    }
</script>
```

コンソールでのメタデータ属性によるフィルタリング

CloudWatch RUM コンソールでビジュアライゼーションを組み込みまたはカスタムのメタデータ属性でフィルタリングするには、検索バーを使用します。検索バーでは、可視化に適用するフィルター用語を [key=value] (キー=値) の形式で最大 20 個指定できます。例えば、Chrome ブラウザのデータのみをフィルタリングするには、browserName=Chrome というフィルター用語を追加できます。

デフォルトでは、CloudWatch RUM コンソールは最も一般的な 100 個の属性のキーと値を取得して、検索バーのドロップダウンに表示します。さらにメタデータ属性をフィルター用語として追加するには、検索バーに属性のキーと値を完全な形で入力します。

フィルターには最大 20 のフィルター用語を含めることができ、アプリケーションモニターごとに最大 20 のフィルターを保存できます。フィルターを保存すると、そのフィルターは [Saved filters] (保存済みのフィルター) ドロップダウンに保存されます。保存したフィルターを削除することもできます。

カスタムイベントを送信する

CloudWatch RUM は、CloudWatch RUM ウェブクライアントによって収集される情報 に記載されているイベントを記録して取り込みます。CloudWatch RUM ウェブクライアントのバージョン 1.12.0 以降を使用している場合は、追加のカスタムイベントを定義、記録、送信できます。定義したイベン

トタイプごとに、イベントタイプ名と送信するデータを定義します。各カスタムイベントペイロードは最大 6 KB です。

カスタムイベントは、アプリケーションモニターでカスタムイベントが有効になっている場合にのみ取り込まれます。アプリケーションモニターの構成設定を更新するには、CloudWatch RUM コンソールまたは UpdateAppMonitor API を使用します。

カスタムイベントを有効にし、カスタムイベントを定義して送信すると、それらを検索できます。それらを検索するには、CloudWatch RUM コンソールの [Events] (イベント) タブを使用します。イベントタイプを使用して検索します。

要件と構文

カスタムイベントは、イベントタイプとイベントの詳細で構成されます。これらに対する要件は次のとおりです。

- イベントタイプ
  - これは、イベントのタイプでも名前でもかまいません。例えば、JsError という名前の CloudWatch RUM 組み込みイベントタイプのイベントタイプは、`com.amazon.rum.js_error_event` です。
  - 1 ~ 256 文字である必要があります。
  - 英数字、アンダースコア、ハイフン、およびピリオドの組み合わせを使用できます。
- イベントの詳細
  - CloudWatch RUM に記録したい実際のデータが含まれます。
  - フィールドと値で構成されるオブジェクトである必要があります。

カスタムイベントの記録例

CloudWatch RUM ウェブクライアントでカスタムイベントを記録するには 2 つの方法があります。

- CloudWatch RUM ウェブクライアントの `recordEvent` API を使用する。
- カスタマイズされたプラグインを使用する。

`recordEvent` API を使用してカスタムイベントを送信する、NPM の例

```
awsRum.recordEvent('my_custom_event', {
        location: 'IAD',
        current_url: 'amazonaws.com',
```

```
        user_interaction: {
            interaction_1 : "click",
            interaction_2 : "scroll"
        },
        visit_count:10
    }
)
```

recordEvent API を使用してカスタムイベントを送信する、埋め込みスクリプトの例

```
cwr('recordEvent', {
    type: 'my_custom_event',
    data: {
        location: 'IAD',
        current_url: 'amazonaws.com',
        user_interaction: {
            interaction_1 : "click",
            interaction_2 : "scroll"
        },
        visit_count:10
    }
})
```

カスタマイズされたプラグインを使用してカスタムイベントを送信する例

```
// Example of a plugin that listens to a scroll event, and
// records a 'custom_scroll_event' that contains the timestamp of the event.
class MyCustomPlugin implements Plugin {
    // Initialize MyCustomPlugin.
    constructor() {
        this.enabled;
        this.context;
        this.id = 'custom_event_plugin';
    }
    // Load MyCustomPlugin.
    load(context) {
        this.context = context;
        this.enable();
    }
    // Turn on MyCustomPlugin.
    enable() {
        this.enabled = true;
        this.addEventHandler();
```

```
    }
    // Turn off MyCustomPlugin.
    disable() {
        this.enabled = false;
        this.removeEventHandler();
    }
    // Return MyCustomPlugin Id.
    getPluginId() {
        return this.id;
    }
    // Record custom event.
    record(data) {
        this.context.record('custom_scroll_event', data);
    }
    // EventHandler.
    private eventHandler = (scrollEvent: Event) => {
        this.record({timestamp: Date.now()})
    }
    // Attach an eventHandler to scroll event.
    private addEventHandler(): void {
        window.addEventListener('scroll', this.eventHandler);
    }
    // Detach eventHandler from scroll event.
    private removeEventHandler(): void {
        window.removeEventListender('scroll', this.eventHandler);
    }
}
```

CloudWatch RUM ダッシュボードの表示

CloudWatch RUM は、ページのロード時間、Apdex スコア、使用されたブラウザとデバイス、ユーザーセッションが置かれた地域的な位置、エラーのあるセッションなど、アプリケーションのパフォーマンスに関するデータをユーザーセッションから収集する際に役立ちます。これらの情報はすべて、ダッシュボードに表示されます。

RUM ダッシュボードを表示するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで、[Application Signals]、[RUM] の順に選択します。

   作成したアプリケーションモニターによって収集された情報は、[Overview] (概要) タブに表示されます。

ペインの一番上の行には、このアプリケーションモニターに関する次の情報が表示されます。

- 実行されたページロードの数
- ページロードの平均速度
- Apdex スコア
- アプリケーションモニターに関連付けられているアラームのステータス

Apdex (Application Performance Index) スコアでは、エンドユーザーの満足度が示されます。このスコアは、0 (最低の満足度) から 1 (最高の満足度) までの範囲を取ります。このスコアは、アプリケーションのパフォーマンスのみを基に算出されています。ユーザーに、アプリケーションの評価が求められることはありません。Apdex スコアの詳細については、「CloudWatch RUM での Apdex スコアの評価方式」を参照してください。

これらのペインの一部には、データを詳しく調べるために使用できるリンクが含まれています。このリンクのいずれかを選択すると、詳細に関するビューが開きます。この画面の上部には、[Performance] 、[Errors] 、[HTTP requests]、[Sessions]、[Events Browsers & Devices] 、[User Journey] などのタブが表示されます。

3. さらに絞り込むには、[List view] (リストビュー) タブを選択し、注目するアプリケーションモニターの名前を選択します。これにより、選択したアプリモニターの次のタブが表示されます。

   - [Performance] (パフォーマンス) タブには、ロード時間、セッション情報、リクエストの情報、ウェブバイタル、ページロードの時間的推移など、ページのパフォーマンスに関する情報が表示されます。このビューにあるコントロールにより、フォーカスする表示内容を [Page loads] (ページロード)、[Requests] (リクエスト)、および [Location] (ロケーション) の間で切り替えることができます。
   - [Errors] タブでは、Javascript エラーに関する情報が表示されます。この情報には、ユーザーに対し最も頻繁に表示されているエラーメッセージ、最もエラーが多いデバイスとブラウザ情報などが含まれています。このビューには、エラーのヒストグラムとエラーのリストビューが含まれます。エラーのリストは、ユーザーとイベントの詳細でフィルタリングできます。エラーメッセージを選択すると、詳細が表示されます。
   - [HTTP requests] タブには、エラーが最も多いリクエスト URL、エラーが最も多いデバイスとブラウザ情報が表示されます。このタブには、リクエストのヒストグラム、リクエストのリストビュー、およびネットワークエラーのリストビューが含まれます。エラーのリストは、ユーザーとイベントの詳細でフィルタリングできます。レスポンスコードまたはエラーメッセージを選択すると、それぞれリクエストまたはネットワークエラーの詳細が表示されます。

- [Sessions] タブには、セッションメトリクスが表示されます。You can request a quota increaseこのタブには、セッション開始イベントのヒストグラムとセッションのリストビューが含まれます。セッションのリストは、イベントタイプ、ユーザーの詳細、およびイベントの詳細でフィルタリングできます。sessionId を選択すると、セッションの詳細が表示されます。
- [Events] タブには、RUM イベントのヒストグラムとイベントのリストビューが表示されます。セッションのリストは、イベントタイプ、ユーザーの詳細、およびイベントの詳細でフィルタリングできます。RUM イベントを選択すると、未処理のイベントが表示されます。
- [Browsers & Devices] (ブラウザとデバイス) タブには、アプリケーションにアクセスした各種のブラウザやデバイスでの、パフォーマンスや使用状況などに関する情報が表示されます。このビューにあるコントロールでは、フォーカスするビューを [ブラウザ] と[デバイス] の間で切り替えます。

  対象を単一のブラウザに絞り込むと、そのブラウザのバージョン別に分類されたデータが表示されます。
- [User Journey] (ユーザージャーニー) タブには、ユーザーがアプリケーションをナビゲートするために使用しているパスが表示されます。ユーザーがアプリケーションに入ったポイントと、アプリケーションを終了したページを確認できます。また、それらのユーザーが通過したパスと、同じパスをたどるユーザーの割合なども確認できます。いずれかのノードで一時停止することで、そのページの詳細を取得できます。1 つのパスを選択すると、接続を見やすく強調表示することもできます。

4. (オプション) 最初の 6 つのタブで [Pages] ボタンを選択すると、リストからページまたはページグループを選択できます。これにより、表示されるデータがアプリケーションの単一ページまたはページグループに関するものに絞り込まれます。さらに、リスト内のページとページグループをお気に入りとしてマークすることも可能です。

CloudWatch RUM での Apdex スコアの評価方式

Apdex (Application Performance Index) は、レポートやベンチマークのための方法、およびアプリケーションの応答時間を評価する方法を定義した、オープンな規格です。Apdex スコアは、時間の経過に伴いアプリケーションのパフォーマンスが受ける影響を特定し、理解するために役立ちます。

Apdex スコアでは、エンドユーザーの満足度が、0 (最低の満足度) から 1 (最高の満足度) のスコア範囲で表されます。このスコアは、アプリケーションのパフォーマンスのみを基に算出されています。ユーザーに、アプリケーションの評価が求められることはありません。

個々の Apdex スコアには、3 つのしきい値のいずれかが対応しています。Apdex のしきい値と、実際のアプリケーションの応答時間に基づいて、パフォーマンスが以下の 3 種類に定義されます。

- 満足 – 実際のアプリケーションの応答時間が、Apdex のしきい値以下。CloudWatch RUM の場合、このしきい値は 2,000 ミリ秒以下です。
- 許容範囲 – 実際のアプリケーションの応答時間は、Apdex のしきい値より長いものの、そのしきい値の 4 倍以下の範囲に収まる。CloudWatch RUM の場合、この範囲は 2,000 ~ 8,000 ミリ秒です。
- 快適ではない – 実際のアプリケーションの応答時間が、Apdex のしきい値の 4 倍を超過。CloudWatch RUM の場合、8000 ミリ秒を超えた範囲がこれに相当します。

最終的な 0 ~ 1 の Apdex スコアは、次の式を使用して計算されます。

```
(positive scores + tolerable scores/2)/total scores * 100
```

CloudWatch RUM で収集できる CloudWatch メトリクス

このセクションの表は、CloudWatch RUM で自動的に収集するメトリクスの一覧です。これらのメトリクスは CloudWatch コンソールで確認できます。詳細については、「利用可能なメトリクスを表示する」を参照してください。

必要に応じて、CloudWatch または CloudWatch Evidently に拡張メトリクスを送信することもできます。詳細については、「拡張メトリクス」を参照してください。

これらのメトリクスは、AWS/RUM という名前の名前空間で発行されます。次のすべてのメトリクスは、application_name ディメンションによって発行されます。このディメンションの値は、アプリケーションモニターの名前です。表に記載されているように、一部のメトリクスは追加のディメンションによって発行されます。

メトリクス	単位	説明
HttpStatusCodeCount	カウント	レスポンスステータスコードによるアプリケーション内の HTTP レスポンスの数。 追加のディメンション:

メトリクス	単位	説明
		• event_details.response.status は、200、400、404 などのレスポンスステータスコードです。 • event_type イベントのタイプ。現在、このディメンションで有効な値は http のみです。
Http4xxCount	カウント	4xx レスポンスステータスコードによるアプリケーション内の HTTP レスポンスの数。 これらは、4xx コードを生成する http_event RUM イベントに基づいて計算されます。

メトリクス	単位	説明
Http5xxCount	カウント	5xx レスポンスステータスコードによるアプリケーション内の HTTP レスポンスの数。 これらは、5xx コードを生成する http_event RUM イベントに基づいて計算されます。
JsErrorCount	カウント	取り込まれた JavaScript エラーイベントの数。
NavigationFrustratedCount	カウント	適切ではないしきい値である 8000ms よりも多い duration でのナビゲーションイベントの数。ナビゲーションイベントの期間は、PerformanceNavigationDuration メトリクスで追跡されます。

メトリクス	単位	説明
NavigationSatisfiedCount	カウント	Apdex の目標である 2000ms 以下の duration でのナビゲーションイベントの数。ナビゲーションイベントの期間は、PerformanceNavigationDuration メトリクスで追跡されます。
NavigationToleratedCount	カウント	2000 ミリ秒から 8000 ミリ秒の間の duration でのナビゲーションイベントの数。ナビゲーションイベントの期間は、PerformanceNavigationDuration メトリクスで追跡されます。
PageViewCount	カウント	アプリケーションモニターによって取り込まれたページビューイベントの数。 これは page_view_event RUM イベントをカウントして計算されます。

メトリクス	単位	説明
PerformanceResourceDuration	ミリ秒	リソースイベントの duration。 追加のディメンション: • event_details.file.type は、スタイルシート、ドキュメント、イメージ、スクリプト、フォントなど、リソースイベントのファイルタイプです。 • event_type イベントのタイプ。現在、このディメンションで有効な値は resource のみです。
PerformanceNavigationDuration	ミリ秒	ナビゲーションイベントの duration。

メトリクス	単位	説明
RumEventPayloadSize	バイト	CloudWatch RUM によって取り込まれるすべてのイベントのサイズ。このメトリクスの SampleCount 統計を使用して、アプリケーションモニターが取り込むイベントの数をモニターリングすることもできます。
SessionCount	カウント	アプリケーションモニターによって取り込まれたセッション開始イベントの数。つまり、開始された新しいセッションの数です。
WebVitalsCumulativeLayoutShift	なし	累積的レイアウトシフトイベントの値を追跡します。
WebVitalsFirstInputDelay	ミリ秒	最初の入力遅延イベントの値を追跡します。
WebVitalsLargestContentfulPaint	ミリ秒	最大のコンテンツフルペイントイベントの値を追跡します。

CloudWatch と CloudWatch Evidently に送信できるカスタムメトリクスと拡張メトリクス

デフォルトでは、RUM アプリケーションモニタはメトリクスを CloudWatch に送信します。これらのデフォルトのメトリクスとディメンションは、「CloudWatch RUM で収集できる CloudWatch メトリクス」に記載されています。

アプリモニターをセットアップして、メトリクスをエクスポートすることもできます。アプリモニターは、拡張メトリクス、カスタムメトリクス、またはその両方を送信できます。それらを CloudWatch または CloudWatch Evident に送信することも、あるいはその両方に送信することもできます。

- カスタムメトリクス — カスタムメトリクスはユーザーが定義するメトリクスです。カスタムメトリクスでは、任意のメトリクス名と名前空間を使用できます。メトリクスを取得するには、任意のカスタムイベント、組み込みイベント、カスタム属性、デフォルト属性を使用します。

  カスタムメトリクスは、CloudWatch と CloudWatch Evidently の両方に送信できます。
- 拡張メトリクス – デフォルトの CloudWatch RUM メトリクスを CloudWatch Evidently に送信し、Evidently の実験に使用できます。追加のディメンションを使用して、デフォルトの CloudWatch RUM メトリクスのいずれかを CloudWatch に送信することもできます。これにより、これらのメトリクスはよりきめ細かいビューを提供します。

トピック

カスタムメトリクス

カスタムメトリクスを送信するには、コンソールではなく、AWS API または AWS CLI を使用する必要があります。AWS API を使用する方法については、「PutRumMetricsDestination」および「BatchCreateRumMetricDefinitions」を参照してください。

1 つの送信先に含めることができる拡張メトリクスおよびカスタムメトリクス定義の最大数は、2,000 です。各送信先に送信するカスタムメトリクスまたは拡張メトリクスごとに、ディメンション名とディメンション値の各組み合わせが、この制限に対してカウントされます。これは、料金計算において CloudWatch カスタムメトリクスとしてもカウントされます。

次の例は、カスタムイベントから派生したカスタムメトリクスの作成方法を示しています。使用されるカスタムイベントの例は次のとおりです。

```
cwr('recordEvent', {
    type: 'my_custom_event',
    data: {
        location: 'IAD',
        current_url: 'amazonaws.com',
        user_interaction: {
            interaction_1 : "click",
            interaction_2 : "scroll"
        },
        visit_count:10
    }
})
```

このカスタムイベントがあれば、Chrome ブラウザから amazonaws.com URL への訪問数をカウントするカスタムメトリクスを作成できます。次の定義では、アカウントの RUM/CustomMetrics/PageVisits 名前空間に AmazonVisitsCount という名前のメトリクスを作成します。

```
{
    "AppMonitorName":"customer-appMonitor-name",
    "Destination":"CloudWatch",
    "MetricDefinitions":[
        {
            "Name":"AmazonVisitsCount",
            "Namespace":"PageVisit",
            "ValueKey":"event_details.visit_count",
            "UnitLabel":"Count",
            "DimensionKeys":{
                "event_details.current_url": "URL"
            },
            "EventPattern":"{\"metadata\":{\"browserName\":[\"Chrome\"]},\"event_type
\":[\"my_custom_event\"],\"event_details\": {\"current_url\": [\"amazonaws.com\"]}}"
        }
    ]
}
```

拡張メトリクス

拡張メトリクスをセットアップすると、次のいずれかまたは両方を行うことができます。

- デフォルトの CloudWatch RUM メトリクスを CloudWatch Evidently に送信し、Evidently の実験に使用します。Evidently に送信できるメトリクスは、PerformanceNavigationDuration、PerformanceResourceDuration、WebVitalsCumulativeLayoutShift、および WebVitalsLargestContentfulPaint のみです。
- ディメンションを追加して、デフォルトの CloudWatch RUM メトリクスのいずれかを CloudWatch に送信すると、メトリクスはよりきめ細かいビューを提供します。例えば、ユーザーが使用する特定のブラウザ固有のメトリクスや、特定の地域にいるユーザーのメトリクスを表示できます。

デフォルトの CloudWatch RUM メトリクスの詳細については、「CloudWatch RUM で収集できる CloudWatch メトリクス」を参照してください。

1 つの送信先に含めることができる拡張メトリクスおよびカスタムメトリクス定義の最大数は、2,000 です。各送信先に送信する拡張メトリクスまたはカスタムメトリクスごとに、ディメンション名とディメンション値の各組み合わせが、この制限に対する拡張メトリクスとしてカウントされます。これは、料金計算において CloudWatch カスタムメトリクスとしてもカウントされます。

CloudWatch に拡張メトリクスを送信すると、CloudWatch RUM コンソールを使用してそのメトリクスに CloudWatch アラームを作成できます。

拡張メトリクスは、CloudWatch カスタムメトリクスとして課金されます。詳細については、Amazon CloudWatch 料金を参照してください。

アプリケーションモニターが送信できるすべてのメトリクス名の拡張メトリクスでは、次のディメンションがサポートされています。これらのメトリクス名は、「CloudWatch RUM で収集できる CloudWatch メトリクス」に記載されています。

- `BrowserName`

  ディメンション値の例: `Chrome`、`Firefox`、`Chrome Headless`
- `CountryCode` これには、2 文字のコードの ISO-3166 形式を使用します。

  ディメンション値の例: `US`、`JP`、`DE`
- `DeviceType`

  ディメンション値の例: `desktop`、`mobile`、`tablet`、`embedded`
- `FileType`

ディメンション値の例: `Image`、`Stylesheet`

- `OSName`

  ディメンション値の例: `Linux`、`Windows`, `iOS`、`Android`

- `PageId`

コンソールを使用して拡張メトリクスをセットアップする

コンソールを使用して CloudWatch に拡張メトリクスを送信するには、次の手順に従います。

CloudWatch Evidently に拡張メトリクスを送信するには、コンソールではなく、AWS API か AWS CLI を使用する必要があります。AWS API を使用して CloudWatch または Evidently に拡張メトリクスを送信する方法については、「PutRumMetricsDestination」と「BatchCreateRumMetricDefinitions」を参照してください。

コンソールを使用してアプリケーションモニタをセットアップし、RUM 拡張メトリクスを CloudWatch に送信するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで、[Application Signals]、[RUM] の順に選択します。
3. [List view] (一覧表示) を選択し、メトリクスを送信するアプリケーションモニターの名前を選択します。
4. [Configuration] (設定) タブを選択し、[RUM extended metrics] (RUM 拡張メトリクス) を選択します。
5. [Send metrics] (メトリクスを送信) を選択します。
6. メトリクス名を 1 つまたは複数選択して、ディメンションを追加して送信します。
7. これらのメトリクスのディメンションとして使用するファクターを 1 つまたは複数選択します。選択すると、その選択によって作成される拡張メトリクスの数が [Number of extended metrics] (拡張メトリクスの数) に表示されます。

   この数は、選択したメトリクス名の数に、作成したさまざまなディメンションの数を掛けて計算されます。この数は、課金されるカスタムメトリクスの数を表します。CloudWatch の料金の詳細については、Amazon CloudWatch の料金をご覧ください。

   a. ページ ID をディメンションとして含むメトリクスを送信するには、[Browse for page ID] (ページ ID を参照) を選択し、使用するページ ID を選択します。

b. デバイスタイプをディメンションとして含むメトリクスを送信するには、[Desktop devices] (デスクトップデバイス) または [Mobile and tablets] (モバイルとタブレット) を選択します。

c. オペレーティングシステムをディメンションとして含むメトリクスを送信するには、[Operating system] (オペレーティングシステム) で 1 つまたは複数のオペレーティングシステムを選択します。

d. ブラウザタイプをディメンションとして含むメトリクスを送信するには、[Browsers] (ブラウザ) で 1 つまたは複数のブラウザを選択します。

e. 地理的位置情報をディメンションとして含むメトリクスを送信するには、[Locations] (ロケーション) で 1 つまたは複数のロケーションを選択します。

このアプリケーションモニターがメトリクスを報告したロケーションのみがリストに表示され、そこから選択できます。

8. 選択が終わったら、[Send metrics] (メトリクスを送信) を選択します。

9. (オプション) [Extended metrics] (拡張メトリクス) リストで、いずれかのメトリクスを監視するアラームを作成するには、そのメトリクスの行で [Create alarm] (アラームを作成) を選択します。

CloudWatch アラーム全般については、「Amazon CloudWatch でのアラームの使用」を参照してください。CloudWatch RUM 拡張メトリクスにアラームを設定するチュートリアルについては、「チュートリアル: 拡張メトリクスを作成してアラームを設定する」を参照してください。

拡張メトリクスの送信を停止する

コンソールを使用して拡張メトリクスの送信を停止するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで、[Application Signals]、[RUM] の順に選択します。
3. [List view] (一覧表示) を選択し、メトリクスを送信するアプリケーションモニターの名前を選択します。
4. [Configuration] (設定) タブを選択し、[RUM extended metrics] (RUM 拡張メトリクス) を選択します。
5. 送信を停止するには、メトリクス名とディメンションの組み合わせを 1 つまたは複数選択します。次に、[Actions] (アクション)、[Delete] (削除) の順に選択します。

チュートリアル: 拡張メトリクスを作成してアラームを設定する

このチュートリアルでは、CloudWatch に送信する拡張メトリクスを設定する方法と、そのメトリクスにアラームを設定する方法を示します。このチュートリアルでは、Chrome ブラウザの JavaScript エラーを追跡するメトリクスを作成します。

この拡張メトリクスをセットアップしてアラームを設定するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで、[Application Signals]、[RUM] の順に選択します。
3. [List view] (一覧表示) を選択し、メトリクスを送信するアプリケーションモニターの名前を選択します。
4. [Configuration] (設定) タブを選択し、[RUM extended metrics] (RUM 拡張メトリクス) を選択します。
5. [Send metrics] (メトリクスを送信) を選択します。
6. [JSErrorCount] を選択します。
7. [Browsers] (ブラウザ) で [Chrome] を選択します。

   この [JSErrorCount] と [Chrome] の組み合わせにより、1 つの拡張メトリクスが CloudWatch に送信されます。このメトリクスでは、Chrome ブラウザを使用するユーザーセッションの JavaScript エラーのみがカウントされます。メトリクス名は [JsErrorCount] になり、ディメンション名は [Browser] (ブラウザ) になります。
8. [Send metrics] (メトリクスを送信) を選択します。
9. [Extended metrics] (拡張メトリクス) リストで、[Name] (名前) に [JsErrorCount] と表示され、[BrowserName] に [Chrome]」と表示されている行で [Create alarm] (アラームを作成) を選択します。
10. [Specify metric and conditions] (メトリクスと条件の指定) で、[Metric name] (メトリクス名) と [BrowserName] フィールドに正しい値が事前に入力されていることを確認します。
11. [Statistic] (統計) で、アラームに使用したい統計を選択します。このタイプのカウントメトリクスには、[Average] (平均) が適しています。
12. [Period (期間)] で、[5 minutes (5 分)] を選択します。
13. [条件] で、次の操作を行います。

    - [Static] (静的) を選択します。
    - エラー数がこれから指定するしきい値を超えるとアラームが ALARM 状態になるように指定するには、[Greater] (より大きい) を選択します。

- [than...] (しきい値) に、アラームしきい値となる数を入力します。5 分間のエラー数がこの数を超えると、アラームは ALARM 状態になります。

14. (オプション) デフォルトでは、エラー数が 5 分間に設定したしきい値を超えるとすぐに、アラームは ALARM 状態になります。必要に応じてこれを変更して、5 分間にこの数を超えることが複数回あった場合に、その回数によってアラームが ALARM 状態になるようにできます。

    そのためには、[Additional configuration] (その他の設定) を選択し、[Datapoints to alarm] (アラームを実行するデータポイント) で、5 分間にエラー数がしきい値を超えることが何回あるとアラームがトリガーされるかを指定します。例えば、2 回中 2 回を選択して 5 分の期間が 2 回連続してしきい値を超えている場合にのみアラームがトリガーされるようにしたり、3 回中 2 回を選択して連続する 3 回の 5 分の期間のうちいずれか 2 回しきい値を超えた場合にアラームがトリガーされるようにしたりできます。

    この種のアラーム評価の詳細については、「アラームの評価」を参照してください。

15. [Next] を選択します。
16. [Configure actions] (アクションの設定) で、アラームがアラーム状態になったときに実行する処理を指定します。Amazon SNS を使用して通知を受け取るには、以下を実行します。

    - 通知を追加 をクリックします。
    - [アラーム状態] を選択します。
    - 既存の SNS トピックを選択するか、新しく作成することができます。新しく作成する場合は、名前を指定し、メールアドレスを少なくとも 1 つ追加します。

17. [Next] を選択します。
18. 名前を入力し、必要に応じてアラームの説明を入力して [Next] (次へ) を選択します。
19. 詳細を確認し、[Create alarm] (アラームを作成) を選択します。

CloudWatch RUM によるデータ保護とデータプライバシー

Amazon CloudWatch RUM でのデータ保護とデータプライバシーには、AWS 責任共有モデルが適用されます。このモデルで定義されるように、AWS は、すべての AWS クラウドを実行するグローバルなインフラストラクチャの保護に責任を負います。お客様は、このインフラストラクチャでホストされているコンテンツに対する管理を維持する責任があります。データプライバシーの詳細については、「データプライバシーのよくある質問」を参照してください。欧州でのデータ保護の詳細については、AWS セキュリティブログの記事「 The AWS Shared Responsibility Model and GDPR」を

参照してください。GDPR 要件の遵守に関するその他のリソースについては、一般データ保護規則 (GDPR) センターを参照してください。

Amazon CloudWatch RUM は、収集対象となるエンドユーザーからのデータ入力に基づいてコードスニペットを生成します。このスニペットは、ウェブサイトまたはウェブアプリケーションのコードに埋め込んで使用します。コードスニペットによってダウンロードおよび構成されたウェブクライアントでは、Cookie (もしくは類似の技術) を利用して、エンドユーザーからのデータを収集できます。Cookie (または類似の技術) の使用は、特定の法領域におけるデータプライバシー規制の対象となります。Amazon CloudWatch RUM を使用する前に、プライバシーに関する法的に適切な通知の提供、クッキーの使用とエンドユーザーデータの (収集を含む) 処理に必要な同意の獲得を含む、適用法の下でのコンプライアンス義務に関して評価を行うことを強くお勧めします。ウェブクライアントが Cookie (または類似の技術) を使用する方法と、ウェブクライアントが収集するエンドユーザーデータの内容については、「CloudWatch RUM ウェブクライアントによって収集される情報」および「CloudWatch RUM ウェブクライアント Cookie (または類似の技術)」を参照してください。

エンドユーザーのアカウント番号、E メールアドレス、その他の個人情報などの機密性がある識別情報は、自由形式のフィールドに格納することは避けるように強くお勧めします。Amazon CloudWatch RUM、または他のサービスに入力したデータは、診断ログに含まれる場合があります。

CloudWatch RUM ウェブクライアント Cookie (または類似の技術)

CloudWatch RUM ウェブクライアントでは、デフォルトで、ユーザーセッションから一定のデータが収集されます。ページロード全体で永続化するユーザー ID とセッション ID をウェブクライアントに収集させるために、Cookie を有効化することができます。ユーザー ID は、RUM によってランダムに生成されます。

これらの Cookie を有効化して、RUM ダッシュボードでアプリケーションモニターに関する表示を行うと、RUM は以下の種類のデータを表示します。

- 一意のユーザー数や、個々のエラーが発生しているユーザーの数など、ユーザー ID に基づいて集計されたデータ。
- セッションの総数やエラーが発生したセッションの数など、セッション ID に基づいて集計されたデータ。
- ユーザージャーニー。これは、サンプリングされた各ユーザーセッションに含まれるページ移動のシーケンスです。

> **Important**
>
> これらの Cookie (または類似の技術) を有効にしない場合でも、ウェブクライアントは、エンドユーザーセッションに関する特定の情報 (ブラウザの種類/バージョン、オペレーティングシステムの種類/バージョン、デバイスの種類など) を記録します。これら収集された情報は、ウェブバイタル、ページの表示数、エラーが発生したページ数など、ページ固有に集約されたインサイトを提供するために使用されます。データ記録の詳細については、「CloudWatch RUM ウェブクライアントによって収集される情報」を参照してください。

CloudWatch RUM ウェブクライアントによって収集される情報

このセクションでは、PutRumEvents スキーマについて記述します。このスキーマでは、CloudWatch RUM を使用してユーザーセッションから収集できるデータの構造を定義します。

PutRumEvents リクエストは、以下のフィールドを持つデータ構造を CloudWatch RUM に送信します。

- この RUM イベントのバッチ ID
- 以下を含む、アプリケーションモニターの詳細情報:
  - アプリケーションモニター ID
  - モニターリング対象アプリケーションのバージョン
- 以下を含むユーザーの詳細情報: このデータは、アプリケーションモニターで Cookie が有効化されている場合にのみ収集されます。
  - ウェブクライアントによって生成されたユーザー ID
  - セッション ID
- このバッチ内での RUM イベントの配列

RUM イベントのスキーマ

各 RUM イベントの構造には、以下のフィールドが含まれます。

- イベントの ID
- タイムスタンプ
- イベントタイプ
- ユーザーエージェント

- Metadata
- RUM イベントの詳細

RUM イベントのメタデータ

メタデータには、ページメタデータ、ユーザエージェントのメタデータ、地理的位置情報のメタデータ、およびドメインメタデータが含まれます。

ページメタデータ

ページメタデータには、以下が含まれます。

- ページ ID
- ページタイトル
- 親ページ ID – これらのデータは、アプリケーションモニタで Cookie が有効化されている場合にのみ収集されます。
- 操作に関する深度 – これは、アプリケーションモニタで Cookie が有効化されている場合にのみ収集されます。
- ページタグ — ページイベントにタグを追加して、ページをグループ化できます。詳細については、「ページグループを使用する」を参照してください。

ユーザーエージェントのメタデータ

ユーザーエージェントのメタデータには、以下が含まれます。

- ブラウザで使用する言語
- ブラウザ名
- ブラウザのバージョン
- オペレーティングシステム名
- オペレーティングシステムのバージョン
- デバイスタイプ
- プラットフォームの種類

地理的位置情報のメタデータ

地理的位置情報のメタデータには以下が含まれます。

- 国コード
- 区域コード

ドメインメタデータ

ドメインメタデータには URL ドメインが含まれます。

RUM イベントの詳細

イベントの詳細は、イベントタイプに応じて、以下のいずれかのタイプのスキーマに従います。

セッション開始イベント

このイベントにはフィールドは含まれません。このデータは、アプリケーションモニターで Cookie が有効化されている場合にのみ収集されます。

ページビュースキーマ

ページビューイベントには、以下のプロパティが含まれます。ページビューコレクションを非アクティブ化するように、ウェブクライアントを設定できます。詳細については、「CloudWatch RUM web client documentation」を参照してください。

名前	型	説明
ページ ID	文字列	アプリケーション内でこのページを一意に表す ID です。デフォルトでは URL パスが使用されます。
親ページ ID	文字列	現在のページに移動する時点で、ユーザーに対し表示されていたページの ID です。このデータは、アプリケーションモニターで Cookie が有効化されている場合にのみ収集されます。
操作に関する深度	文字列	このデータは、アプリケーションモニターで Cookie が有効化されている場合にのみ収集されます。

JavaScript エラースキーマ

エージェントによって生成される JavaScript エラーイベントには、以下のプロパティが含まれます。ウェブクライアントは、エラーテレメトリの収集を選択した場合にのみ、これらのイベントを収集します。

名前	型	説明
エラータイプ	文字列	エラーの名前 (存在する場合)。詳細については、「Error.prototype.name」を参照してください。 ブラウザによっては、エラータイプをサポートしていない場合があります。
エラーメッセージ	文字列	エラーが返すメッセージ。詳細については、「Error.prototype.message」を参照してください。エラーフィールドが存在しない場合、これはエラーイベントのメッセージになります。詳細については、「ErrorEvent」を参照してください。 異なるブラウザ間では、エラーメッセージが一貫しない場合があります。
スタックトレース	文字列	エラーのスタックトレース (それが存在する場合) は 150 文字に切り捨てられます。詳細については、「Error.prototype.stack」を参照してください。 ブラウザによっては、スタックトレースをサポートしていない場合があります。

DOM イベントスキーマ

エージェントによって生成されるドキュメントオブジェクトモデル (DOM) イベントには、以下のプロパティが含まれます。これらのイベントは、デフォルトでは収集されません。収集されるのは、インタラクションテレメトリをアクティブにした場合のみです。詳細については、「CloudWatch RUM web client documentation」を参照してください。

名前	型	説明
イベント	文字列	クリック、スクロール、ホバーなど、DOM イベントの種類。詳細については、「Event reference」を参照してください。
要素	文字列	DOM 要素タイプ

名前	型	説明
要素 ID	文字列	イベントを生成した要素が ID を持つ場合、その ID がこのプロパティに格納されます。詳細については、「Element.id」を参照してください。
CSSLocator	文字列	DOM 要素の識別に使用される CSS ロケーター。
InteractionId	文字列	ユーザーと UI 間のインタラクションにおける一意の ID です。

ナビゲーションイベントスキーマ

ナビゲーションイベントは、アプリケーションモニターでパフォーマンステレメトリが有効になっている場合にのみ収集されます。

ナビゲーションイベントでは、ナビゲーションタイミングレベル 1、およびナビゲーションタイミングレベル 2 の API を使用します。レベル 2 の API は、すべてのブラウザーでサポートされている訳ではないため、これらの新しいフィールドはオプションです。

> **Note**
>
> タイムスタンプのメトリクスは、DOMHighResTimestamp に基づいています。レベル 2 の API のデフォルトでは、すべてのタイミングが `startTime` に対し相対的です。また、レベル 1 では、タイムスタンプメトリクスから `navigationStart` メトリクスを減算して相対値を算出します。タイムスタンプの値はすべてミリ秒単位で表示されます。

ナビゲーションイベントには、以下のプロパティが含まれます。

名前	型	説明	メモ
initiatorType	文字列	パフォーマンスイベントを開始したリソースのタイプを表します。	値:「ナビゲーション」 レベル 1:「ナビゲーション」

名前	型	説明	メモ
			レベル 2: entryData .initiatorType

名前	型	説明	メモ
navigationType	文字列	ナビゲーションのタイプを表します。 この属性は必須ではありません。	値: この値は以下のいずれかにする必要があります。 • navigate は、リンクの選択、ブラウザのアドレスバーへの URL の入力、フォームの送信、または (reload または back_forward 以外の) スクリプト処理からの初期化によって開始されるナビゲーションです。 • reload は、ブラウザのリロード処理もしくは location.reload() によるナビ

名前	型	説明	メモ
			ゲーションです。 • back_forward は、ブラウザでの履歴横断の処理によるナビゲーションです。 • prerender は、プリレンダリングヒントによって開始されるナビゲーションです。詳細については、「Prerender」を参照してください。

名前	型	説明	メモ
startTime	数	イベントがトリガーされるタイミングを示します。	値: 0 レベル 1: entryData.navigationStart - entryData.navigationStart レベル 2: entryData.startTime

名前	型	説明	メモ
unloadEventStart	数	ウィンドウ内の前のドキュメントが、アンロードを (unload イベントがスローされた後に) 開始したタイミングを示します。	値:先に表示されたドキュメントが存在しないか、そのドキュメントもしくは必要なリダイレクトの 1 つでオリジンが異なっている場合、返される値は 0 です。 レベル 1: `entryData.unloadEventStart > 0 ? entryData.unloadEventStart - entryData.navigationStart : 0` レベル 2: entryData.unloadEventStart

名前	型	説明	メモ
promptForUnload	数	ドキュメントのアンロードに要した時間。つまり unloadEventStart と unloadEventEnd の間の時間です。UnloadEventEnd は、アンロードイベントのハンドラが終了したタイミングを、ミリ秒単位の時間で表します。	値:先に表示されたドキュメントが存在しないか、そのドキュメントもしくは必要なリダイレクトの 1 つでオリジンが異なっている場合、返される値は 0 です。 レベル 1: entryData.unloadEventEnd - entryData.unloadEventStart レベル 2: entryData.unloadEventEnd - entryData.unloadEventStart

名前	型	説明	メモ
redirectCount	数	現在のブラウジングコンテキストの下で行われた、最後の非リダイレクトナビゲーション以降のリダイレクトの数を表す数値。 この属性は必須ではありません。	値:リダイレクトが存在しない場合、またはリダイレクトがあってもそのオリジンが送信先ドキュメントと異なっている場合、返される値は 0 です。 レベル 1: 利用不可 レベル 2: entryData.redirectCount

名前	型	説明	メモ
redirectStart	数	最初の HTTP リダイレクトが開始された時刻。	値:リダイレクトが存在しない場合、またはリダイレクトがあってもそのオリジンが送信先ドキュメントと異なっている場合、返される値は 0 です。 レベル 1: `entryData.redirectStart > 0 ? entryData.redirectStart - entryData.navigationStart : 0` レベル 2: entryData .redirectStart

名前	型	説明	メモ
redirectTime	数	HTTP リダイレクトに要した時間。これは、`redirectStart` と `redirectEnd` の間の差です。	レベル 1:: entryData .redirectEnd - entryData .redirectStart レベル 2:: entryData .redirectEnd - entryData .redirectStart

名前	型	説明	メモ
workerStart	数	これは PerformanceResourceTiming インターフェイスのプロパティです。この値は、ワーカースレッドでのオペレーションの開始を示します。 この属性は必須ではありません。	値: Service Worker スレッドがすでに実行中である、またはそのスレッドを開始する直前である場合、このプロパティは FetchEvent をディスパッチする直前の時間を返します。リソースが Service Worker によって傍受されない場合は 0 を返します。 レベル 1: 利用不可 レベル 2: entryData .workerStart

名前	型	説明	メモ
workerTime	数	リソースが Service Worker によって傍受されている場合、この値にはワーカースレッドでの処理に要する時間が返されます。 この属性は必須ではありません。	レベル 1: 利用不可 レベル 2: `entryData.workerStart > 0 ? entryData.fetchStart - entryData.workerStart : 0`
fetchStart	数	ブラウザで、HTTP リクエストによるドキュメントのフェッチが準備ができた時刻。これは、アプリケーションキャッシュがチェックされる前のタイミングです。	レベル 1: `: entryData.fetchStart > 0 ? entryData.fetchStart - entryData.navigationStart : 0` レベル 2: entryData.fetchStart

名前	型	説明	メモ
domainLookupStart	数	ドメインルックアップが開始された時間。	値: 永続接続が使用されている場合、または情報がキャッシュまたはローカルリソースに格納されている場合、この値は `fetchStart` と同じ値になります。 レベル 1: `entryData.domainLookupStart > 0 ? entryData.domainLookupStart - entryData.navigationStart : 0` レベル 2: entryData.domainLookupStart

名前	型	説明	メモ
dns	数	ドメインルックアップに必要な時間。	値: リソースと DNS レコードがキャッシュされている場合、この値には 0 が想定されます。 レベル 1: entryData.domainLookupEnd - entryData.domainLookupStart レベル 2: entryData.domainLookupEnd - entryData.domainLookupStart
nextHopProtocol	文字列	リソースの取得に使用されるネットワークプロトコルを表す文字列。 この属性は必須ではありません。	レベル 1: 利用不可 レベル 2: entryData.nextHopProtocol

名前	型	説明	メモ
connectStart	数	ユーザーエージェントがドキュメントを取得するためにサーバーへの接続を確立した時点の直前の時間。	値: RFC2616 の永続接続が使用されている場合、または現在のドキュメントが関連するアプリケーションキャッシュまたはローカルリソースから取得された場合、この属性は値として domainLookupEnd を返します。 レベル 1: `entryData.connectStart > 0 ? entryData.connectStart - entryData.navigationStart : 0` レベル 2: entryData.connectStart

名前	型	説明	メモ
接続	数	転送のための接続の確立、または SSL 認証の実行に必要な時間を測定します。また、ブラウザによって発行される同時リクエストが多すぎる場合の、ブロック時間の長さも含まれます。	レベル 1: entryData .connectEnd - entryData .connectStart レベル 2: entryData .connectEnd - entryData .connectStart
secureConnectionStart	数	現在のページの URL スキームが「https」の場合、この属性は、ユーザーエージェントが現在の接続を保護するためのハンドシェイクプロセスを開始する直前の時間を返します。HTTPS を使用しない場合は 0 を返します。URL スキームの詳細については、「URL representation」を参照してください。	計算式: entryData .secureConnectionStart

名前	型	説明	メモ
tlsTime	数	SSL ハンドシェイクを完了するのに要した時間。	レベル 1: `entryData.secureConnectionStart > 0 ? entryData.connectEnd - entryData.secureConnectionStart : 0` レベル 2: `entryData.secureConnectionStart > 0 ? entryData.connectEnd - entryData.secureConnectionStart : 0`

名前	型	説明	メモ
requestStart	数	ユーザーエージェントが、サーバー、関連するアプリケーションキャッシュ、またはローカルからのリソースに対する要求を開始した時点の直前の時間。	レベル 1: `: entryData.requestStart > 0 ? entryData.requestStart - entryData.navigationStart : 0` レベル 2: entryData .requestStart
timeToFirstByte	数	リクエストが発行された後、情報の最初のバイトを受信するまでに要した時間。この時間は `startTime` に対して相対的です。	レベル 1: entryData .responseStart - entryData .requestStart レベル 2: entryData .responseStart - entryData .requestStart

名前	型	説明	メモ
responseStart	数	ユーザーエージェントの HTTP パーサーが、関連するアプリケーションキャッシュ、ローカルリソース、またはサーバーから、最初の応答バイトを受信した直後の時間。	レベル 1: `entryData.responseStart > 0 ? entryData.responseStart - entryData.navigationStart : 0` レベル 2: entryData.responseStart

名前	型	説明	メモ
responseTime	文字列	関連するアプリケーションキャッシュ、ローカルリソース、またはサーバーから、バイト形式の完全な応答を受信するまでに要した時間。	レベル 1: `entryData.responseStart > 0 ? entryData.responseEnd - entryData.responseStart : 0` レベル 2: `entryData.responseStart > 0 ? entryData.responseEnd - entryData.responseStart : 0`

名前	型	説明	メモ
domIntera ctive	数	パーサーがメインドキュメントでの作業を終了し、HTML DOM が構築された時間。この際には、`Document.readyState` が「interactive」に変化し、それに対応する `readystatechange` イベントがスローされます。	レベル 1: `entryData.domInteractive > 0 ? entryData.domInteractive - entryData.navigationStart : 0` レベル 2: entryData .domInter active

名前	型	説明	メモ
domContentLoadedEventStart	数	ユーザーエージェントが、現在のドキュメントで DOMContentLoaded イベントを発行した時点の直前の時間を表します。DOMContentLoaded イベントは、最初の HTML ドキュメントが完全に読み込まれ、解析されたタイミングで発行されます。この時点では、メイン HTML ドキュメントの解析が完了し、ブラウザがレンダーツリーの作成を開始しており、また、サブリソースをロードする必要があります。これにより、スタイルシート、画像、およびサブフレームのローディングが完了するのを待つことはありません。	レベル 1: `entryData.domContentLoadedEventStart > 0 ? entryData.domContentLoadedEventStart - entryData.navigationStart : 0` レベル 2: entryData.domContentLoadedEventStart

名前	型	説明	メモ
domContentLoaded	数	これは、レンダーツリー構築の開始と終了の時間で、domContentLoadedEventStart と domContentLoadedEventEnd により マークされます。この値により、CloudWatch RUM は実行の追跡を行います。このプロパティ値は、domContentLoadedStart と domContentLoadedEnd の間の差です。 この期間の間に、DOM と CSSOM の準備が整います。このプロパティは、非同期的および動的に作成されたスクリプトを除き、スクリプトが実行されるまで待機します。また、スクリプトがスタイルシートに依存している場合、domContentLoaded はスタイルシートのために待機します。画像のためには待機しません。 **Note** domContentLoadedStart そして domContentLoadedEnd での実際の値は、Google Chrome の ネットワークパネルに表示される domContentLoaded に近似します。この値は、ページの読み込みプロセスの開始から、HTML DOM と CSSOM のレンダリングツリーが構築されるまでに要した時間を示します。ナビゲーションメトリクスの場合、domContentLoaded 値は開始値と終了値間の差を表します。これは、サブリソースのダウンロードとレンダリングツリー構築に要した時間のみを反映します。	レベル 2: entryData.domContentLoadedEventEnd – entryData.domContentLoadedEventStart レベル 2: entryData.domContentLoadedEventEnd – entryData.domContentLoadedEventStart

名前	型	説明	メモ
domComplete	数	現在のドキュメントの準備を完了したことを、ブラウザが知らせてきた時点の直前の時間。この時点で、画像などのサブリソースの読み込みは完了しています。これには、CSS や同期 JavaScript などのブロッキングコンテンツのダウンロードにかかる時間も含まれます。この値は、Google Chrome のネットワークパネルに表示される `loadTime` の値に近似します。	レベル 1: `entryData.domComplete > 0 ? entryData.domComplete - entryData.navigationStart : 0` レベル 2: entryData.domComplete
domProcessingTime	数	レスポンスとロードイベント開始までの間の合計時間。	レベル 1: entryData.loadEventStart - entryData.responseEnd レベル 2: entryData.loadEventStart - entryData.responseEnd

名前	型	説明	メモ
loadEvent Start	数	現在のドキュメントの load イベントが発行された直前の時間。	レベル 1: `entryData .loadEvent tStart > 0 ? entryData .loadEvent tStart - entryData .navigati onStart : 0` レベル 2: entryData .loadEvent tStart
loadEvent Time	数	loadEventStart と loadEventEnd の違い。このロードイベントを待っている追加の関数またはロジックは、この間に起動されます。	レベル 1: entryData .loadEvent tEnd - entryData .loadEvent tStart レベル 2: entryData .loadEvent tEnd - entryData .loadEvent tStart

名前	型	説明	メモ
duration	文字列	この期間は、ページの合計読み込み時間です。メインページとそのすべての同期サブリソースをダウンロードし、さらにページをレンダリングするため要した時間を記録します。この後も、スクリプトなどの非同期リソースのダウンロードは継続します。この値は、`loadEventEnd` と `startTime` の 2 つのプロパティ間の差になります。	レベル 1: entryData.loadEventEnd - entryData.navigationStart レベル 2: entryData.duration
headerSize	数	`transferSize` と `encodedBodySize` の間の差を返します。 この属性は必須ではありません。	レベル 1: 利用不可 レベル 2: entryData.transferSize – entryData.encodedBodySize レベル 2: entryData.transferSize – entryData.encodedBodySize

名前	型	説明	メモ
compressionRatio	数	encodedBodySize と decodedBodySize の比率です。encodedBodySize の値は、HTTP ヘッダーを除くリソースの圧縮後のサイズです。decodedBodySize の値は、HTTP ヘッダーを除くリソースの解凍後のサイズです。 この属性は必須ではありません。	レベル 1: 利用不可。 レベル 2: `entryData.encodedBodySize > 0 ? entryData.decodedBodySize / entryData.encodedBodySize : 0`
navigationTimingLevel	数	ナビゲーションタイミング API のバージョン。	値: 1 または 2

リソースイベントスキーマ

リソースイベントは、アプリケーションモニターでパフォーマンステレメトリが有効化されている場合にのみ収集されます。

タイムスタンプのメトリクスは、型定義 DOMHighResTimeStamp に基づきます。レベル 2 の API のデフォルトでは、すべてのタイミングは startTime に対して相対的です。レベル 1 API の場合には、タイムスタンプのメトリクスから navigationStart メトリクスが減算され、相対値が取得されます。タイムスタンプの値はすべてミリ秒単位で表示されます。

エージェントによって生成されたリソースイベントには、以下のプロパティが含まれます。

名前	型	説明	メモ
targetUrl	文字列	リソースの URL を返します。	計算式: entryData .name
initiatorType	文字列	パフォーマンスリソースイベントを開始したリソースのタイプを表します。	値: 「resource」 計算式: entryData .initiatorType
duration	文字列	`responseEnd` と `startTime` の 2 つのプロパティ間の差を返します。 この属性は必須ではありません。	計算式: entryData .duration
transferSize	数	レスポンスヘッダーフィールドとレスポンスペイロード本体を含む、フェッチされたリソースのサイズ (オクテット単位) を返します。 この属性は必須ではありません。	計算式: entryData .transferSize
fileType	文字列	ターゲット URL パターンから派生した拡張。	

最大のコンテンツフルペイントイベントのスキーマ

最大のコンテンツペイントイベントには、以下のプロパティが含まれています。

これらのイベントは、アプリケーションモニターでパフォーマンステレメトリが有効化されている場合にのみ収集されます。

名前	説明		
値	詳細については、「Web		

名前	説明		
	Vitals」を参照してください。		

最初の入力遅延イベント

最初の入力遅延イベントには、以下のプロパティが含まれます。

これらのイベントは、アプリケーションモニターでパフォーマンステレメトリが有効化されている場合にのみ収集されます。

名前	説明		
値	詳細については、「Web Vitals」を参照してください。		

累積的レイアウトシフトイベント

累積的レイアウトシフトイベントには、以下のプロパティが含まれます。

これらのイベントは、アプリケーションモニターでパフォーマンステレメトリが有効化されている場合にのみ収集されます。

名前	説明		
値	詳細については、「Web Vitals」を参照してください。		

HTTP イベント

HTTP イベントには、以下のプロパティが含まれます。これには、`Response` フィールドまたは `Error` フィールドのいずれかが含まれますが、両方が含まれることはありません。

これらのイベントは、アプリケーションモニターで HTTP テレメトリが有効化されている場合にのみ収集されます。

名前	説明
リクエスト	リクエストフィールドには以下が含まれます。 • GET、POST などの値を含めることができる `Method` フィールドです。 • URL
レスポンス	レスポンスには以下が含まれます。 • 2xx、4xx、または 5xx などのステータス • ステータスのテキスト
エラー	このエラーフィールドには以下が含まれます。 • タイプ • メッセージ • ファイル名 • 行番号 • 列番号 • スタックトレース

X-Ray トレースイベントのスキーマ

これらのイベントは、アプリケーションモニターで X-Ray トレースが有効化されている場合にのみ収集されます。

X-Ray トレースイベントのスキーマの詳細については、AWS X-Ray segment documents を参照してください。

単一ページアプリケーションのルート変更の時間計測

従来のマルチページアプリケーションでは、ユーザーが新しいコンテンツのロードを要求すると、ユーザーは実際にはサーバーから新しい HTML ページを要求することになります。その結

果、CloudWatch RUM ウェブクライアントは、通常のパフォーマンス API メトリクスを使用してロード時間をキャプチャします。

しかし、単一ページのウェブアプリケーションでは、JavaScript と Ajax を使用して、サーバーから新しいページをロードせずにインターフェイスが更新されます。単一ページの更新は、ブラウザの時間計測 API では記録されず、ルート変更の時間計測を使用します。

CloudWatch RUM は、サーバーからの全ページロードと単一ページ更新の両方のモニターリングをサポートしていますが、次の違いがあります。

- ルート変更の時間計測については、`tlsTime`、`timeToFirstByte` などのようなブラウザ提供のメトリクスはありません。
- ルート変更の時間計測については、`initiatorType` フィールドは `route_change` になります。

CloudWatch RUM ウェブクライアントは、ルートの変更につながる可能性のあるユーザーインタラクションをリッスンし、そのようなユーザーインタラクションが記録されると、ウェブクライアントはタイムスタンプを記録します。次の両方に該当すると、ルート変更の時間計測が開始されます。

- ブラウザ履歴 API (ブラウザの [進む] ボタンと [戻る] ボタンを除く) を使用してルート変更を実行した。
- ルート変更検出時刻と最新のユーザーインタラクションタイムスタンプの差が 1000 ms 未満。これにより、データスキューが回避されます。

その後、ルート変更の時間計測が始まると、進行中の AJAX リクエストと DOM ミューテーションがない場合、その時間計測は完了します。このとき、最後に完了したアクティビティのタイムスタンプが完了タイムスタンプとして使用されます。

進行中の AJAX リクエストまたは DOM ミューテーションが 10 秒 (デフォルト) を超えると、ルート変更の時間計測がタイムアウトします。この場合、CloudWatch RUM ウェブクライアントは、このルート変更の時間計測を記録しなくなります。

その結果、ルート変更イベントの期間は次のように計算されます。

```
(time of latest completed activity) - (latest user interaction timestamp)
```

CloudWatch RUM を使用するアプリケーションを管理する

下記のセクションの各ステップを使用して、アプリケーションでの CloudWatch RUM の使用を管理します。

既に生成してあるコードスニペットを見つけるにはどうすればよいですか?

既にアプリケーション用に生成してある CloudWatch RUM コードスニペットを検索するには、以下のステップを実行します。

既に生成したコードスニペットを検索するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで、[Application Signals]、[RUM] の順に選択します。
3. [List view] (一覧表示) をクリックします。
4. アプリケーションモニターの名前の横で、[View JavaScript] (JavaScript を表示) をクリックします。
5. [JavaScript Snippet] (JavaScript スニペット) ペインで、[Copy to clipboard] (クリップボードにコピー) をクリックします。

アプリケーションを編集する

アプリケーションモニターの設定を変更するには、以下のステップに従います。アプリケーションモニターでは、名前以外の任意の設定の変更が可能です。

アプリケーションでの CloudWatch RUM の使用方法を編集するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで、[Application Signals]、[RUM] の順に選択します。
3. [List view] (一覧表示) をクリックします。
4. アプリケーションの名前の横にあるボタンをクリックした後、 [Actions] (アクション)、[Edit] (編集) の順にクリックします。
5. アプリケーション名以外の設定を変更します。この設定の詳細については、「ステップ 2: アプリケーションモニターを作成する」を参照してください。
6. 完了したら、[Save] (保存) を選択します。

設定を変更することで、コードスニペットも変更されます。この場合は、更新されたコードスニペットをアプリケーション内に挿入しなおす必要があります。

7. JavaScript のコードスニペットが作成されたら、[Copy to clipboard] (クリップボードにコピー) または[Download] (ダウンロード) を選択した上で、[Done] (完了) をクリックします。

   新しい設定でモニターリングを開始するには、新しいコードスニペットをアプリケーションに挿入します。コードスニペットを、アプリケーションの (<body> 要素または他のすべての <script> タグの前で) <head> 要素の中に挿入します。

CloudWatch RUM の使用を停止するかアプリケーションモニターを削除する

アプリケーションでの CloudWatch RUM の使用を停止するには、アプリケーションのコードから、RAM が生成したコードスニペットを削除します。

RUM のアプリケーションモニターを削除するには、以下のステップに従います。

アプリケーションモニターを削除するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで、[Application Signals]、[RUM] の順に選択します。
3. [List view] (一覧表示) をクリックします。
4. アプリケーションの名前の横にあるボタンをクリックし、[Actions] (アクション)、[Delete] (削除) の順にクリックします。
5. 確認ボックスに **Delete** を入力し、[Delete] (削除) をクリックします。
6. まだ残されているのであれば、アプリケーションのコードから CloudWatch RUM コードスニペットを削除します。

CloudWatch RUM でのクォータ

CloudWatch RUM には、以下のクォータがあります。

リソース	デフォルトのクォータ
アプリケーションモニター数	アカウントあたり 20 個

リソース	デフォルトのクォータ
	クォータは、引き上げをリクエストすることができます。
RUM 取り込み率	50 PutRumEvents リクエスト/秒 (TPS)。 クォータは、引き上げをリクエストすることができます。

CloudWatch RUM のトラブルシューティング

このセクションでは、CloudWatch RUM でのトラブルシューティングに役立つヒントを示します。

実行しているアプリケーションに関するデータが見つからない

まず、コードスニペットがアプリケーションに正しく挿入されていることを確認します。詳細については、「ステップ 4: コードスニペットをアプリケーションに挿入する」を参照してください。

それが原因ではない場合は、アプリケーションへのトラフィックがまだ発生していないことが考えられます。ユーザーと同じ方法でアプリケーションにアクセスし、トラフィックを生成して確認ます。

アプリケーションのデータの記録が停止する

アプリケーションが更新され、CloudWatch RUM のコードスニペットが含まれなくなった可能性があります。アプリケーションコードを確認します。

他には、誰かがコードスニペットを更新した後に、更新されたスニペットをアプリケーションに挿入しなかった可能性もあります。既に生成してあるコードスニペットを見つけるにはどうすればよいですか? にある指示に従って、現時点の正しいコードスニペットを見つけます。その上で、アプリケーションに貼り付けられているコードスニペットと比較します。

ネットワークモニタリング

このセクションのトピックでは、AWS リソースに関するオペレーションの可視性を高めるのに役立つモニタ機能を提供する CloudWatch モニタリングについて説明します。

トピック

Amazon CloudWatch Internet Monitor の使用

Amazon CloudWatch Internet Monitor は、AWS でホストされているアプリケーションとエンドユーザーの間で、インターネットの問題がパフォーマンスや可用性にどのように影響しているかを可視化します。これにより、インターネットの問題の診断にかかる時間が、数日から数分に短縮できます。Internet Monitor では、AWS のグローバルネットワークのフットプリントから接続データを取得します。そして、インターネット向けトラフィックのパフォーマンスと可用性に関するベースラインの計算に使用します。このデータは、インターネットの稼働時間と可用性を監視するために AWS が使用するものと同じデータです。ベースラインであるこれらの測定値を基に、アプリケーションを実行しているさまざまな地理的位置のエンドユーザー (クライアント) に重大な問題が発生した際、Internet Monitor はそれを識別し通知を行います。

Amazon CloudWatch コンソールでは、トラフィックパターンとヘルスイベントの全体像を確認できます。このイベントに関する情報は、さまざまな地域的詳細度 (ロケーション) で容易に掘り下げることができます。影響を明確に可視化し、その影響を受けているクライアントのロケーションとネットワーク (ASN、通常はインターネットサービスプロバイダー (ISP)) を特定できます。Internet Monitor は、特定の ASN または AWS ネットワークが原因で、インターネットのアベイラビリティまたはパフォーマンスの問題が発生していると判断した場合、その情報を提供します。

Internet Monitor の主な機能

- Internet Monitor は、エンドユーザーのエクスペリエンスを向上させるために役立つインサイトと推奨事項を提案します。アプリケーションの予測されるレイテンシーをどのように改善するか、ほぼリアルタイムで探索できます。そのために、使用するサービスの切り替えや、さまざまな AWS リージョン を経由してのワークロードに対するトラフィックの再ルーティングをします。
- Internet Monitor を使用すると、アプリケーションのパフォーマンスと可用性に影響を与えている要因をすばやく特定し、問題を追跡して対処できます。

- Internet Monitor は、インターネット測定値を CloudWatch Logs と CloudWatch メトリクスにパブリッシュします。この測定値と、アプリケーションに固有の口ケーションや ASN (インターネットサービスプロバイダー) に関するヘルス情報で、CloudWatch ツールの使用をサポートします。必要に応じて、インターネット測定値を Amazon S3 にパブリッシュすることもできます。
- Internet Monitor のヘルスイベントは Amazon EventBridge に送信されるので、これを使用して通知を設定できます。問題の原因が AWS ネットワークである場合は、この問題を軽減するために AWS が講じている手段が記載された AWS Health Dashboard 通知も自動的に送信されます。

Internet Monitor の使用方法

Internet Monitor を使用するには、モニタを作成し、アプリケーションのリソースをそのモニタ、VPC、Network Load Balancers、CloudFront ディストリビューション、または WorkSpaces ディレクトリに関連付けます。これにより、Internet Monitor はアプリケーションのインターネット向けトラフィックの場所を認識します。その後、Internet Monitor は AWS からインターネット測定値を発行します。この測定値は都市ネットワーク、つまりクライアントがアプリケーションにアクセスするクライアントの口ケーションと ASN (通常はインターネットサービスプロバイダー (ISP)) に対して固有のものです。詳細については、「Amazon CloudWatch Internet Monitor の仕組み」を参照してください。Internet Monitor を使い始めるには、「コンソールを使用して Amazon CloudWatch Internet Monitor の使用を開始する」を参照してください。

内容

- CloudWatch ログとメトリクスのアクセスエラーのトラブルシューティング
- Amazon CloudWatch Internet Monitor でのデータ保護とデータプライバシー
- Amazon CloudWatch Internet Monitor 用 Identity and Access Management
- Amazon CloudWatch Internet Monitor のクォータ

Amazon CloudWatch Internet Monitor がサポートされる AWS リージョン

このセクションには、Amazon CloudWatch Internet Monitor がサポートされる AWS リージョン がリストされています。Internet Monitor がサポートされるリージョン (オプトインリージョンを含む) の最新リストについては、「Amazon Web Services 全般のリファレンス」の「Amazon CloudWatch Internet Monitor のエンドポイントとクォータ」を参照してください。

Internet Monitor は、モニタを作成した AWS リージョン にのみモニターのデータを保存しますが、1 つのモニタに複数のリージョンのリソースを含めることができます。

リージョン名 (オプトインサポート)	リージョン
アフリカ (ケープタウン)	af-south-1
アジアパシフィック (香港)	ap-east-1
アジアパシフィック (ハイデラバード)	ap-south-2
アジアパシフィック (ジャカルタ)	ap-southeast-3
アジアパシフィック (メルボルン)	ap-southeast-4
欧州 (ミラノ)	eu-south-1
欧州 (スペイン)	eu-south-2
欧州 (チューリッヒ)	eu-central-2
中東 (バーレーン)	me-south-1
中東 (アラブ首長国連邦)	me-central-1

リージョン名 (デフォルトサポート)	リージョン
米国東部 (オハイオ)	us-east-2
米国東部 (バージニア北部)	us-east-1
米国西部 (北カリフォルニア)	us-west-1
米国西部 (オレゴン)	us-west-2
アジアパシフィック (ムンバイ)	ap-south-1
アジアパシフィック (大阪)	ap-northeast-3
アジアパシフィック (ソウル)	ap-northeast-2
アジアパシフィック (シンガポール)	ap-southeast-1
アジアパシフィック (シドニー)	ap-southeast-2
アジアパシフィック (東京)	ap-northeast-1
カナダ (中部)	ca-central-1
欧州 (フランクフルト)	eu-central-1
欧州 (アイルランド)	eu-west-1
欧州 (ロンドン)	eu-west-2
欧州 (パリ)	eu-west-3
欧州 (ストックホルム)	eu-north-1
南米 (サンパウロ)	sa-east-1

Amazon CloudWatch Internet Monitor の料金

Amazon CloudWatch Internet Monitor には前払いコストや長期間のコミットメントはありません。Internet Monitor には、モニタリング対象リソースごとと都市ネットワークごとの 2 つの料金体

系があります。都市ネットワークとは、クライアントがアプリケーションリソースにアクセスするロケーションと、そのアクセスのために経由するネットワーク (ASN (インターネットサービスプロバイダー (ISP) など)) のことです。ログや作成した追加のメトリクス、ダッシュボード、アラーム、インサイトについても、標準の CloudWatch 料金が請求されることに注意してください。

モニタを作成する際には、監視するトラフィックの割合を選択します。モニタリングする都市ネットワークの最大数に上限を設定して、請求額を抑えることができます。モニタを編集すれば、いつでも、監視するトラフィックの割合や都市ネットワークの最大数を変更することができます。最初は (アカウントごとの全モニタで) 100 の都市ネットワークが含まれます。それ以降は、実際に追加されたモニタリングする都市ネットワークの数に対してのみ、最大数まで課金されます。

モニタリングする実際に追加された都市ネットワークの数 (最大数まで) に対してのみ請求されます。アカウントごとの全モニタで最初の 100 の都市ネットワークについては、料金は発生しません。100 の都市ネットワークの費用に相当する一律の金額が、毎月の請求額から差し引かれます。

例えば、グローバルな大企業であれば、インターネット向けトラフィックの 100% を監視して、リソースが 1 つのモニタリング 1 つに対しての最大ネットワーク数を 50,000 に設定することができます。トラフィックが 50,000 の都市ネットワークに達したとすると、その部分の請求額は月額約 2,700 USD になります。特定の地理的なエリアにおいて、1 つのモニターに 1 つのリソースと 200 の都市ネットワークを備えている別の企業の場合、この部分の請求額は月額約 13 USD/月になります。詳細については、「都市ネットワークの上限を選択する」を参照してください。

料金見積りツールを使うと、さまざまなオプションで見積ることができます。料金オプションを確認するには、CloudWatch の料金見積りツールのページで「Internet Monitor」まで下へスクロールします。

Internet Monitor と CloudWatch の料金の詳細については、「Amazon CloudWatch の料金」を参照してください。

Amazon CloudWatch Internet Monitor のコンポーネントと用語

Amazon CloudWatch Internet Monitor は以下を使用または参照します。

モニター

モニタには、インターネットのパフォーマンスおよび可用性の測定値の表示や、ヘルスイベントアラートの受け取りが必要な単一アプリケーションのリソースが含まれます。アプリケーションのモニタを作成する際は、アプリケーションのリソースを追加して、Internet Monitor がモニタリングする都市 (ロケーション) を定義します。Internet Monitor は、ユーザーが追加したアプリケーションリソースからのトラフィックパターンを使用して、アプリケーションと通信している口

ケーションと ASN (通常はインターネットサービスプロバイダー (ISP)) に固有のインターネットのパフォーマンスおよび可用性に関する測定値をパブリッシュします。つまり、追加したリソースによって、Internet Monitor がモニタリングし、測定結果をパブリッシュする都市ネットワークのスコープが決まります。

モニタリングに追加されたリソース (「モニタリング対象リソース」)

モニタに追加するリソースは、Internet Monitor のモニタリング対象リソースです。つまり、次のようになります。

- リージョンに追加する各 VPC はモニタリング対象リソースです。VPC を追加すると、Internet Monitor はその VPC 内のインターネット向けアプリケーションのトラフィックをすべてモニタリングします。例えば、Amazon EC2 インスタンスでホストされているアプリケーション、Network Load Balancer の背後にあるアプリケーション、AWS Fargate コンテナなどです。
- リージョンに追加する各 Network Load Balancer はモニタリング対象リソースです。
- リージョンに追加する各 WorkSpaces ディレクトリはモニタリング対象リソースです。
- 追加する各 CloudFront ディストリビューションはモニタリング対象リソースです。

AS 番号 (ASN)

通常、Internet Monitor での ASN は、Verizon や Comcast といったインターネットサービスプロバイダー (ISP) を意味します。ASN は、クライアントがインターネットアプリケーションにアクセスするために使用するネットワークプロバイダーです。AS (自律システム) とは、単一のネットワーク、またはネットワークが形成する集合に属する、インターネットでルーティング可能なインターネットプロトコル (IP) プレフィックスのセットのことです。これらはすべて、1 つの組織によって管理、制御、監視されます。

都市ネットワーク (ロケーションと ASN)

都市ネットワークとは、クライアントがアプリケーションリソースにアクセスする (都市などの) ロケーションと、そのアクセスのために経由する ASN (通常はインターネットサービスプロバイダー (ISP)) のことです。Internet Monitor がモニタリングする都市ネットワークの最大数をモニタごとに上限設定すると、請求額を抑えることができます。課金されるのは、実際にモニタリングする都市ネットワークの最大数までです。詳細については、「都市ネットワークの最大数の選択」を参照してください。

インターネット測定値

Internet Monitor は、インターネット測定値を 5 分ごとに CloudWatch Logs のログファイルにパブリッシュします。対象は、アカウント内の上位 500 の都市ネットワーク (クライアントロ

ケーションと ASN、通常はインターネットサービスプロバイダー (ISP)) です。これらの測定値は、アプリケーションの都市ネットワークのパフォーマンススコア、可用性スコア、転送バイト数 (受信バイト数と送信バイト数)、往復時間を定量化します。これらは VPC、Network Load Balancers、CloudFront ディストリビューション、または WorkSpaces ディレクトリに固有の都市ネットワークの測定値です。必要に応じて、モニタリング対象の都市ネットワークすべてのインターネット測定値とイベントを Amazon S3 バケットにパブリッシュする選択ができます (ただし、サービスの上限数は 500,000 の都市ネットワークです)。

メトリクス

Internet Monitor は、アプリケーションへのグローバルトラフィックと、各 AWS リージョン へのグローバルトラフィックについて、CloudWatch メトリクス用に集約したメトリクスを生成します。詳細については、「Amazon CloudWatch Internet Monitor での CloudWatch Metrics の使用」を参照してください。

ヘルスイベント

Internet Monitor はヘルスイベントを作成し、アプリケーションに影響を与える特定の問題に関するアラートをユーザーに提供します。Internet Monitor は、世界中のネットワーク遅延の増加などのインターネットの問題を検出します。次に、AWS のグローバルインフラストラクチャのフットプリント全体から収集したインターネット測定値の履歴を使用して、現在の問題がアプリケーションに与える影響を計算し、ヘルスイベントを作成します。Internet Monitorは、デフォルトで、全体的な影響と局所的な影響のしきい値の両方に基づいてヘルスイベントを作成します。しきい値の設定について詳しくは、「ヘルスイベントのしきい値の変更」を参照してください。

各ヘルスイベントには、影響を受けている都市ネットワークに関する情報が含まれています。ヘルスイベントは CloudWatch コンソールで表示するか、AWS SDK を使用、または AWS CLI で Internet Monitor API アクションを使用することで表示できます。Internet Monitor は、ヘルスイベントのために Amazon EventBridge 通知も送信します。詳細については、「Internet Monitor がヘルスイベントを作成して解決するタイミング」を参照してください。

インターネットイベント

Internet Monitor では、インターネットイベントと呼ばれる最近のグローバルヘルスイベントに関する情報が、AWS のすべてのお客様が利用できるインターネットの状況マップに表示されます。インターネットの状況マップを表示するには、Internet Monitor でモニターを作成する必要はありません。ヘルスイベントとは異なり、インターネットイベントは個々のお客様やそのアプリケーショントラフィックに限定されません。詳細については、「Amazon CloudWatch Internet Monitor の世界中のインターネットの状況マップ」を参照してください。

しきい値

Internet Monitorは、全体的なしきい値と局所的なしきい値の両方に基づいてヘルスイベントを作成します。デフォルトのしきい値を変更したり、ローカルしきい値をオフにするなど、他のオプションを構成したりできます。しきい値の設定について詳しくは、「ヘルスイベントのしきい値の変更」を参照してください。

パフォーマンスと可用性スコア

Internet Monitor は AWS が収集したデータを分析し、アプリケーションのパフォーマンスや可用性が Internet Monitor が計算した推定のベースラインを下回った際にそれを検出できます。このような情報はスコアの形式で通知されるので、ユーザーは低下が発生したことを容易に確認できます。パフォーマンススコアは、パフォーマンスの低下とは認識されないトラフィックの割合の推定値を表します。同様に可用性スコアでは、可用性の低下とはならないトラフィックの割合の推定値を表します。詳細については、「AWS によるパフォーマンススコアと可用性スコアの計算方法」を参照してください。

転送バイト数とモニタリング対象転送バイト数

転送バイト数は、クライアントがアプリケーションにアクセスする AWS のアプリケーションと都市ネットワーク間で送受信されたトラフィックの総バイト数です (都市ネットワークとは、ロケーションと ASN、通常はインターネットサービスプロバイダーのことです)。モニタリング対象転送バイト数も同様のメトリクスですが、モニタリング対象トラフィックのバイト数のみを含みます。

往復時間

往復時間 (RTT) は、クライアントユーザーのリクエストからユーザーに応答を返すまでにかかる時間を表します。RTT がクライアントのロケーション (都市または他の地域) にわたって集計される場合、その値は、各クライアントのロケーションでどの程度のアプリケーショントラフィックが発生するかによって重み付けされます。

Amazon CloudWatch Internet Monitor の世界中のインターネットの状況マップ

Amazon CloudWatch Internet Monitor には、AWS のすべてのお客様が利用できる世界中のインターネットの状況マップが表示されます。マップを表示するには、Amazon CloudWatch コンソールで [インターネットモニター] に移動します。

このマップでは、パフォーマンスや可用性に問題がある特定の都市やネットワーク (ASN、通常はインターネットサービスプロバイダー) を含め、AWS のお客様に影響を与える世界中のインターネッ

トイベント (「停止」) が強調表示されています。インターネットの状況マップには、過去 24 時間のインターネットイベントが含まれています。

インターネットの状況マップを表示するには、Internet Monitor でモニターを作成する必要はありません。Internet Monitor のヘルスイベントとは異なり、インターネットイベントは個々のお客様やそのアプリケーショントラフィックに限定されません。

インターネットの状況マップでは、インターネットイベントを選択して詳細を確認できます。インターネットイベントの場合、開始時間、終了時間 (イベントが終了した場合)、現在のステータス (アクティブまたは解決済み)、およびシステム停止問題のタイプ (アベイラビリティまたはパフォーマンス) を確認できます。インターネットの状況マップの作成方法と内容の詳細については、世界中のインターネットの状況マップに関する FAQ をご覧ください。

アプリケーショントラフィックとクライアントロケーションに固有の詳細情報を表示して操作するには、Internet Monitor でアプリケーションのモニターを簡単に設定できます。次に、現在と過去におけるパフォーマンスおよび可用性のパターンとイベントが表示されます。また、アプリケーションのフットプリントとお客様に合わせて調整されたヘルスイベントアラートも表示されます。インターネットの状況マップでは全体を見ることができ、特定のモニターでは情報をフィルタリングして、アプリケーションに関連する測定値と詳細のみを表示します。モニターを使用すると、履歴メトリクスを調べたり、アプリケーションのクライアントエクスペリエンスを向上させるための推奨事項を確認したりすることもできます。詳細については、「コンソールを使用して Amazon CloudWatch Internet Monitor の使用を開始する」を参照してください。

Amazon CloudWatch Internet Monitor の仕組み

このセクションでは、Amazon CloudWatch Internet Monitor の仕組みを説明します。この説明には、インターネット全体における接続上の問題を検出するために使用するデータを AWS が収集する方法や、パフォーマンスと可用性のスコアの計算方法に関する説明も含まれます。

目次

- 含める都市ネットワークのサブセットを Internet Monitor が選択する仕組み
- 世界中のインターネットの状況マップを作成する方法 (よくある質問)

Internet Monitor でアプリケーショントラフィックのフットプリントのみに重点をおく方法

Internet Monitor は、AWS リソースのユーザーがアクセスするインターネットについて、そのサブセットだけをモニタリングすることに重点を置いています。この点が、お客様のウェブサイトを世界中のあらゆるリージョンから全体的にモニタリングする他のツールとは異なります。また、このソリューションは費用対効果も高く、大企業だけでなく中小企業にとっても導入しやすくなっています。

Internet Monitor は、AWS が社内的に活用しているものと同じ強力なプローブと問題検出アルゴリズムを使用します。そして、Internet Monitor 内にヘルスイベントを作成し、アプリケーションに影響を与える接続上の問題をお客様にアラートします。その後、Internet Monitor は、アクティブな利用者から作成したトラフィックプロファイルを重ねることで得られた、パフォーマンスと可用性に関する (アプリケーションリソースに基づいた) マップを提供します。

Internet Monitor はこの情報を使用して、関連のあるイベント (つまり、アクティブな利用者がいる場所でのイベント) や、それらのイベントが全体的な利用者数に及ぼす影響を的確に表示します。つまり、イベントが与える影響度は、全世界のトラフィックの総量を基に (割合として) 算出されます。

Internet Monitor は、インターネット測定値を 5 分ごとに CloudWatch Logs にパブリッシュします。対象は、各モニタにトラフィックを送信する上位 500 の都市ネットワーク (クライアントロケーションと ASN、通常はインターネットサービスプロバイダー (ISP)) です。必要に応じて、モニタリング対象の都市ネットワークすべてのインターネット測定値を Amazon S3 バケットにパブリッシュする選択ができます (ただし、サービスの上限数は 500,000 の都市ネットワークです)。詳細については、「Amazon CloudWatch Internet Monitor で Amazon S3 にインターネット測定値をパブリッシュする」を参照してください。

Internet Monitor には、次のような利点があります。

- Internet Monitor の使用が、AWS でホストされているアプリケーションに、追加の負荷やコストをかけることはありません。
- クライアント側のリソースやアプリケーション内に、パフォーマンス測定用のコードを含める必要はありません。
- アプリケーションが接続されているインターネットについて、「ラストマイル」情報を含めたパフォーマンスと可用性の情報に関する可視化が行えます。

Internet Monitor は、AWS のリソースに基づいて測定値を作成するため、作成されるイベントは、すべてアプリケーショントラフィックに固有であることに注意してください。インターネットにおける、通常の一般的な世界規模の問題は報告の対象になりません。また、サービスロケーションが AWS リージョン の場合、生成される測定値やイベントはリージョンレベルでの接続性を表すように設計されたものとなります。つまり、エンドユーザーのロケーションとアベイラビリティーゾーン間の接続性を正確には表しません。

AWS で接続の問題を測定し測定値を計算する方法

Amazon CloudWatch Internet Monitor は、自律システム番号 (ASN) を介して、異なる AWS リージョン と Amazon CloudFront の POP (Point Of Presence) との間のインターネット接続データを、さまざまなクライアントのロケーションに対して使用します。通常はインターネットサービスプロバイダー (ISP) です。これは、世界中のインターネットにおける接続問題を事前に検出するために、AWS のオペレーターが社内で日常的に使用している接続データです。

当社では AWS リージョン のすべてについて、インターネットのどの部分がそのリージョンと通信しているかを把握しており、次のことを行っています。

- 当社は、インターネットのこれらの部分を、30 日間の間隔で繰り返し積極的に監視しています。
- これには、ネットワークのプローブと、より上位のプロトコル用のプローブの両方 (インバウンドとアウトバウンドのプローブなど) を使用します。

AWS は、アクティブとパッシブの 2 種類のプローブを使用して、90 パーセンタイルでのレイテンシー (パフォーマンス) と、すべての AWS リージョン および CloudFront サービスからインターネット全体までの到達可能性 (可用性) を測定しています。利用者の場所とサービスとの間で、接続の異常なパターンを監視し、お客様にアラートを送ります。

可用性と RTT の計算

ラウンドトリップタイム (RTT) は、ユーザーからのリクエストがユーザーに応答を返すまでにかかる時間長を表します。エンドユーザーロケーションの間で往復時間を集計すると、その値は、各エンドユーザーのロケーションによって誘導されるトラフィック量によって、重み付けされます。

例えば、エンドユーザーのロケーションが 2 つあり、その 1 つは 5 ミリ秒の RTT でトラフィックの 90% を処理し、もう 1 つは 10 ミリ秒の RTT でトラフィックの 10% を処理している場合、結果として集計される RTT は 5.5 ミリ秒 (5 ミリ秒× 0.9 + 10 ミリ秒× 0.1) になります。

ラストマイルのレイテンシーの測定に関しては、リソースごとに違いがあることに留意してください。Internet Monitor のレイテンシーの測定では、VPC、Network Load Balancers、および WorkSpaces ディレクトリにはラストマイルのレイテンシーは含まれません。

パフォーマンスと可用性スコアの計算

AWS は、AWS のサービスとさまざまな都市ネットワーク (ロケーションと ASN) の間におけるインターネットのパフォーマンスおよび可用性に関する膨大な履歴データを保持しています。このデータに統計分析を適用することで、アプリケーションのパフォーマンスと可用性が、事前に推定計算してあるベースラインを下回った際に、Internet Monitor はそれを検出できます。これらの低下を確認しやすくするために、その情報はヘルススコア (パフォーマンスと可用性のスコア) の形式でユーザーに報告されます。

ヘルススコアの計算は、さまざまな詳細度で行われます。もっとも高い詳細度では、都市や大都市圏などの地域および ASN (都市ネットワーク) ごとに、ヘルススコアを計算します。また、個々のヘルススコアを、アプリケーションの全体的なヘルススコア番号としてモニター内にまとめます。特定の地域やサービスプロバイダーでフィルタリングせずにパフォーマンススコアや可用性コアを表示する場合、Internet Monitor は全体的なヘルススコアを表示します。

全体的なヘルススコアは、指定された期間でアプリケーション全体をカバーします。アプリケーションの都市ネットワークのペアのパフォーマンスまたは可用性スコアが、パフォーマンスまたは可用性についての対応するヘルスイベントのしきい値にグローバルに達するか、またはそれを下回ると、Internet Monitor はヘルスイベントをトリガーします。デフォルトでは、パフォーマンスと可用性の両方のしきい値は 95% です。Internet Monitor は、オプションがデフォルトで有効になっている場合は、設定した値に基づいてローカルしきい値に基づいてヘルスイベントも作成します。ヘルスイベントのしきい値の変更の詳細については、「ヘルスイベントのしきい値を変更する」を参照してください。

モニターおよびログファイル内の情報を掘り下げて問題を調査し、詳細を確認する際には、特定の都市 (場所)、ネットワーク(ASN またはインターネットサービスプロバイダー)、またはその両方でフィルタリングできます。したがって、フィルターを使用すると、選択したフィルターに応じて、さまざまな都市、ASN、または都市ネットワークのペアのヘルススコアを確認できます。

- 可用性スコアは、トラフィックにおいて可用性の低下だと判定されない割合の推定値を表します。Internet Monitor は、観測されたトラフィックの総量と可用性メトリクスの測定値から、トラフィックが低下した割合を推定します。例えば、エンドユーザーとサービスロケーションのペアにおける可用性スコアが 99% である場合、そのペアでは、トラフィックの 1% で可用性が低下していることになります。

- パフォーマンススコアは、トラフィックにおいてパフォーマンスの低下だと判定されない割合を表します。例えば、エンドユーザーとサービスロケーションのペアにおけるパフォーマンススコアが 99% の場合、そのペアでは、トラフィックの 1% でパフォーマンスが低下していることになります。

TTFB と RTT の計算 (レイテンシー)

最初のバイトまでの時間 (TTFB) は、クライアントがリクエストを実行した後、サーバーからの情報の最初のバイトを受信するまでにかかった時間です。TTFB の AWS の計算では、Amazon EC2 または Amazon CloudFront から Internet Monitor 測定ノードまでの経過時間を測定します (ノードのラストマイルを含む)。つまり、Internet Monitor は、EC2 の TTFB の場合はユーザーから Amazon EC2 リージョンまでの時間を測定し、CloudFront の TTFB の場合はユーザーから CloudFront までの時間を測定します。

ラウンドトリップ時間 (RTT) の場合、Internet Monitor は、パブリック IP アドレスによってマッピングされた都市ネットワーク (つまり、クライアントのロケーションと ASN、通常はインターネットサービスプロバイダー) から AWS リージョン までの時間を考慮します。これは、Internet Monitor が、ゲートウェイまたは VPN の背後からインターネットにアクセスするユーザーのラストマイルを知ることができないことを意味します。

ラストマイルのレイテンシーの測定に関しては、リソースごとに違いがあることに留意してください。Internet Monitor のレイテンシーの測定では、VPC、Network Load Balancers、および WorkSpaces ディレクトリにはラストマイルのレイテンシーは含まれません。

Internet Monitor では、CloudWatch ダッシュボードの [トラフィックインサイト] タブの [トラフィック最適化の提案] セクションに、TTFB の情報が表示されます。これは、パフォーマンスを改善できるアプリケーションのさまざまな設定オプションを評価するのに役立ちます。

リージョンとアベイラビリティーゾーンの測定値と集計

Internet Monitor は測定値を集約し、影響の共有をリージョンレベルで行いますが、影響の計算はアベイラビリティーゾーン (AZ) レベルで行います。つまり、1 回のイベントで 1 つの AZ のみが影響を受け、トラフィックの大部分がその AZ を通過する場合、トラフィックへの影響を確認できます。ただし、同じイベントで、影響を受けた AZ をアプリケーショントラフィックが通貨しない場合、影響は確認できません。

これは WorkSpaces ディレクトリではないリソースにのみ適用されます。WorkSpaces ディレクトリは、リージョンレベルのみで測定されます。

Internet Monitor の位置情報の精度

位置情報については、Internet Monitor は MaxMind から提供される IP 位置情報データを使用します。Internet Monitor の測定における位置情報の精度は、MaxMind のデータの精度に依存します。

`Metro` レベルの測定値は、米国以外の場所では正確ではない可能性があるため注意が必要です。

Internet Monitor がヘルスイベントを作成および解決する場合

Internet Monitor は、現在設定されているしきい値に基づいて、監視対象のアプリケーショントラフィックのヘルスイベントを作成および終了します。Internet Monitor にはデフォルトのしきい値設定があり、独自のしきい値を設定することもできます。Internet Monitor は、接続の問題がアプリケーションに及ぼす全体的な影響と、アプリケーションにクライアントが存在するローカルエリアへの影響を判断し、しきい値を超えるとヘルスイベントを作成します。

Internet Monitor は、AWS で提供されるサービスで利用可能なインターネットのパフォーマンスとネットワークトラフィックの可用性に関する履歴データに基づいて、接続の問題がクライアントロケーションに与える影響を計算します。クライアントがアプリケーションを使用する ASN とサービスの地理的位置、つまり影響を受ける都市とネットワークのペアに基づいて、アプリケーションに関連する情報を適用します。場所は、モニターに追加したリソースによって決まります。次に、Internet Monitor は統計分析を使用して、パフォーマンスと可用性が低下し、アプリケーションのクライアントエクスペリエンスに影響が及ぶことを検出します。

パフォーマンスと可用性のスコアは、低下が見られないトラフィックのパーセンテージとして表されます。影響度合いはこの逆で、お客様のエンドユーザーにとって、この問題がどの程度の割合で障害になっているかを表します。仮に、全世界的に可用性が 93% まで低下している場合には、その影響度は 7% になります。

アプリケーションの都市ネットワークのペアのパフォーマンスまたはアベイラビリティスコアが、パフォーマンスまたはアベイラビリティについての対応するヘルスイベントのしきい値にグローバルに達するか、またはそれを下回ると、Internet Monitor がトリガーされ、ヘルスイベントが生成されます。デフォルトでは、パフォーマンスとアベイラビリティの両方のしきい値は 95% です。しきい値を満たすか、またはしきい値を下回る値は累積されるため、いくつかの小さなイベントが積み重なってしきい値の割合を満たすことがある一方で、単一のイベントがしきい値レベルを満たしたり、しきい値を下回ったりする可能性もあります。

イベントをトリガーしたパフォーマンスまたは可用性スコアが、対応するヘルスイベントのしきい値の割合以下である限り、ヘルスイベントはアクティブなままとなります。イベントをトリガーしたスコアまたは合計スコアがしきい値を超えると、Internet Monitor はヘルスイベントを解決します。

また、Internet Monitor は、ローカルのしきい値と、問題が影響するトラフィック全体の割合に基づいてヘルスイベントを作成します。ローカルなしきい値のオプションを設定することも、ローカルなしきい値を完全に無効にすることもできます。

ヘルスイベントのしきい値の変更の詳細については、「ヘルスイベントのしきい値を変更する」を参照してください。

ヘルスイベントが報告されるタイミング

Internet Monitor には、インターネットの問題に関するすべてのシグナルを収集するための機能が備わっており、それを使用して、ヘルスイベントはモニター内に数分で作成されます。

可能な場合、Internet Monitor はヘルスイベントの発生源を分析して、原因が AWS にあるのか、それとも ASN によるものなのかを判断します。ヘルスイベントの分析は、イベントが解決された後も継続されます。新しい情報があると、Internet Monitor はイベントを最大 1 時間で更新できます。

Internet Monitor が IPv4 および IPv6 トラフィックを処理する仕組み

Internet Monitor は、IPv4 のみを介してネットワークの正常性を測定し、任意の IP ファミリー (IPv4 または IPv6) を介してそのネットワークにトラフィックを提供している場合には、ヘルスイベント、可用性およびパフォーマンスのメトリクスを表示します。デュアルスタック CloudFront ディストリビューションなどのデュアルスタックリソースからのトラフィックを提供する場合、IPv4 トラフィックに IPv6 トラフィックと同じリソースの問題がある場合にのみ、Internet Monitor はヘルスイベントを発生させ、パフォーマンススコアまたは可用性スコアの低下を示します。

全体のバイト (IN) とバイト (OUT) に関する Internet Monitor のメトリクスは、すべてのインターネットトラフィック (IPv4 および IPv6) を正確に反映していることに注意してください。

含める都市ネットワークのサブセットを Internet Monitor が選択する仕組み

モニターでモニタリングする都市ネットワークの上限数を設定すると、または、モニタリングするトラフィックの割合を選択すると、Internet Monitor が、最近の最も高いトラフィック量に基づいて、含める (モニタリングする) 都市ネットワークを選択します。

例えば、都市ネットワークの上限数を 100 に設定すると、Internet Monitor は、最近 1 時間のアプリケーショントラフィックに基づいて、(最大) 100 件の都市ネットワークをモニタリングします。具体的には、Internet Monitor は、直近 1 時間の時間枠より前の 1 時間の時間枠で、最もトラフィックが多かった上位 100 件の都市ネットワークをモニタリングします。

例えば、現在の時刻が午後 2 時 30 分であるとします。その場合、モニターに表示されるトラフィックは午後 1 時から午後 2 時の間にキャプチャされたものであり、Internet Monitor が上位

100 件の都市ネットワークを選択するために用いる、トラフィック量の測定値は、午後 0 時から午後 1 時までの間にキャプチャされたものになります。

世界中のインターネットの状況マップを作成する方法 (よくある質問)

Amazon CloudWatch Internet Monitor のインターネットの状況マップは、認証済みの AWS のすべてのお客様が Internet Monitor コンソールで利用できます。このセクションには、インターネットの状況マップの作成方法と使用方法に関する詳細が含まれています。

Internet Monitor のインターネットの状況マップとは何ですか?

インターネットの状況マップは、世界中のインターネット問題を視覚的に表しています。影響を受けるクライアントロケーション、つまり、都市に ASN (通常はインターネットサービスプロバイダー) を加えたものが強調表示されます。このマップでは、世界中の主要なクライアントロケーションと AWS サービスについて、クライアントのインターネットエクスペリエンスに最近影響を与えた可用性とパフォーマンスの問題の組み合わせを示しています。

マップのデータの提供元はどこですか?

データは、インターネットのアクティブプロービングとパッシブプロービングの組み合わせに基づいています。Internet Monitor がデータを測定する方法の詳細については、「AWS による接続問題の測定方法」のセクションを参照してください。

マップはどのくらいの頻度で更新されますか?

インターネットの状況マップは 15 分ごとに更新されます。

どのネットワークが停止について追跡されていますか?

AWS は、インターネットを AWS に接続するためにお客様が使用する重要な IP プレフィックスを表す世界中のネットワークを追跡します。AWS ネットワークとの間で送受信されるトラフィック量のトップのトーカーであるクライアントロケーションが停止の対象となります。

インターネットイベントがマップに含まれるかどうかは何で決まりますか?

インターネットイベントがインターネットの状況マップに含まれるかどうかを判断するために使用する大まかな基準は次のとおりです。

- AWS は、アベイラビリティイベントまたはパフォーマンスイベントがあることを検出します。
- イベントの有効期間が短い (例えばイベントが 5 分未満の) 場合、そのイベントは無視されます。
- そして、そのイベントがトップのトーカーに分類されるクライアントロケーションで行われる場合、そのイベントは停止とみなされます。

インターネットの状況マップにはどのようなしきい値が使用されますか?

インターネットの状況マップでは、停止を判断するためのしきい値は固定されていません。Internet Monitor は、期待値からの逸脱を検出した結果に基づいて、何がイベントを構成するのかを判断します。この仕組みの詳細については、サービスで作成したモニターのヘルスイベントをいつ作成するかを Internet Monitor が判断する方法をご覧ください。モニターを作成すると、Internet Monitor は独自のアプリケーショントラフィックに固有のインターネットトラフィックヘルス測定値を生成します。また、Internet Monitor は、アプリケーションのインターネットトラフィックに影響する問題に関するヘルスイベントを通知します。

このデータで何ができますか?

インターネットの状況マップには、過去 24 時間に世界中で発生した主要なインターネットイベントの簡単な概要が表示されます。インターネットトラフィックを Internet Monitor にオンボードしなくても、インターネットモニタリングエクスペリエンスのフレーバーを取得するのに役立ちます。AWS のインターネットモニタリング機能を最大限に活用して、AWS にホストされているアプリケーションやサービスに合わせてパーソナライズするには、Internet Monitor でモニターを作成できます。

モニターを作成すると、Internet Monitor を有効にして、アプリケーションクライアントに影響する特定のインターネットパスを識別できるようになり、クライアントエクスペリエンスの向上に役立つ機能を利用できるようになります。また、アプリケーショントラフィックとクライアントに特に影響する新しいインターネットの問題についても、事前に通知されます。

イベントの詳細を知るにはどうすればよいですか?

マップ上の停止をクリックすると、イベントの開始時と終了時、影響を受けた都市と ASN、問題の種類 (つまり、パフォーマンスの問題や可用性の問題) などの詳細が表示されます。

イベントに関する詳細情報を取得したり、アプリケーショントラフィックのカスタム測定値を取得したりするには、Internet Monitor でモニターを作成します。

Amazon CloudWatch Internet Monitor のユースケース例

このセクションでは、いくつかの具体的な例について説明するとともに、詳細を記載したブログ記事へのリンクを示します。これらの例は、アプリケーションをモニタリングし、ユーザーエクスペリエンスを改善するために、Amazon CloudWatch Internet Monitor の機能をどのように役立てることができるのかを示しています。

アラートを設定し、講じるべきアクションを決定する

Internet Monitor を使用すると、時間の経過に伴う平均的なインターネットパフォーマンスメトリクスに関するインサイトや、都市ネットワーク(クライアントのロケーションと ASN、通常はインターネットサービスプロバイダー) ごとのヘルスイベントに関するインサイトを得ることができます。Internet Monitor を使用すると、Amazon Virtual Private Clouds (VPC)、Network Load Balancers、Amazon WorkSpaces、または Amazon CloudFront によってホストされるアプリケーションのエンドユーザーエクスペリエンスに影響を与えているイベントを特定できます。

モニターを作成した後、Internet Monitor のヘルスイベントに関するアラートを受け取る方法について、いくつかのオプションがあります。これらには、ヘルスイベントをフィルタリングするためのイベントメトリクスまたは Amazon EventBridge ルールを使用する、CloudWatch アラームに基づく通知が含まれます。アラームに基づく通知やアクションは、CloudWatch ロググループへの AWS SMS 通知や更新など、さまざまなオプションから選択できます。

例に関して、ガイダンスを含む詳細については、ブログ記事の「Amazon CloudWatch Internet Monitor の紹介」を参照してください。

レイテンシーの問題を特定し、TTFB を改善してマルチプレイヤーゲームプレイエクスペリエンスをより良いものにする

Internet Monitor を使用すると、グローバルクラウドゲームアプリのゲームプレイヤーにレイテンシーの問題が生じている場所を世界中で迅速に特定し、パフォーマンスを向上させるためのインサイトを得ることができます。最も多くのプレイヤーの最初のバイトまでの時間 (TTFB) が最も遅くなっている場所を特定することで、最大のプレイヤーベースをより満足させるためにレイテンシーを改善する方法がわかります。

次に、ゲーム用に次の EC2 サーバーをデプロイする準備ができた時には、Internet Monitor が推奨する AWS リージョン を選択することで、レイテンシが高くプレイヤーのグループが最も大きいエリアの TTFB を低下させることができます。

このユースケースに対して Internet Monitor を設定し使用するための詳細については、ブログ記事の「Amazon CloudWatch Internet Monitor を使用してゲームエクスペリエンスを高める」を参照してください。

Internet Monitor のクロスアカウントオブザーバビリティ

Internet Monitor のクロスアカウントオブザーバビリティを使用すると、単一のAWS リージョン内の AWS アカウントにまたがるアプリケーションをモニタリングできます。

Amazon CloudWatch オブザーバビリティアクセスマネージャーを使用して、1 つ以上の AWS アカウントをモニタリングアカウントとして設定できます。モニタリングアカウントにシンクを作成することで、モニタリングアカウントでソースアカウントのデータを表示できるようになります。シンクは、モニターリングアカウントのアタッチメントポイントを表すリソースです。Internet Monitor では、モニターがリソースのアタッチメントポイントになります。シンクを使用して、ソースアカウントからモニタリングアカウントへのリンクを作成します。詳細については、「CloudWatch のクロスアカウントオブザーバビリティ」を参照してください。

必要なリソース

CloudWatch Application Insights のクロスアカウントオブザーバビリティが適切に機能するためには、以下のテレメトリータイプが CloudWatch Observability Access Manager を通じて共有されていることを確認してください。

- Internet Monitor のモニター
- Amazon CloudWatch のメトリクス
- Amazon CloudWatch Logs のロググループ

コンソールを使用して Amazon CloudWatch Internet Monitor の使用を開始する

Amazon CloudWatch Internet Monitor の使用を開始するには、そのアプリケーションが使用する AWS リソースを追加し、いくつかの構成オプションを設定して、Internet Monitor でアプリケーション用のモニターを作成する必要があります。この章では、コンソールでモニターを追加する手順について説明します。また、Internet Monitor のリソースについて詳しく説明するセクションと、モニター用に設定できる、または設定する必要がある、さまざまなオプションに関する説明と制限事項を含む追加のセクションも含まれています。

内容

- Amazon VPC で Amazon CloudWatch Internet Monitor を追加または作成する
- CloudFront を使用して、Amazon CloudWatch Internet Monitor のモニターを追加または作成する

コンソールを使用して Amazon CloudWatch Internet Monitor でモニターを作成する

Amazon CloudWatch Internet Monitor でアプリケーション用にモニターを作成するには、そのモニターが使用する AWS リソースを追加し、いくつかの構成オプションを設定します。追加したリソースの Amazon Virtual Private Cloud (VPC)、Network Load Balancers (NLB)、CloudFront ディストリビューション、WorkSpaces ディレクトリは、Internet Monitor に情報を提供して、アプリケーションのインターネットトラフィック情報をマッピングします。モニターを作成したら、15 ~ 30 分待機して、アプリケーションが使用されている場所に固有のトラフィックプロファイルを生成します。次に、Internet Monitor のモニターまたは他のツールを使用して、クライアントの使用状況に関するパフォーマンスとアベイラビリティを視覚化し、詳しく確認できます。これらのツールは、モニターによって収集され、CloudWatch Logs などに発行されたアプリケーショントラフィックの測定値を使用してインサイトを提供します。

Internet Monitor では、基本的に 1 つのアプリケーションに 1 つのモニタを作成するのが最も簡単です。同じモニター内で、さまざまな場所や ASN (通常はインターネットサービスプロバイダー)、または他の情報ごとに、Internet Monitor のログファイル内の測定値とメトリクスを検索したり、並べ替えたりできます。さまざまな分野のアプリケーション用に、例えば個別のモニタを作成する必要はありません。

このステップでは、コンソールを使用して、モニタのセットアップを順を追って説明します。モニターの作成やイベントの表示などを実行するために Internet Monitor API アクションと AWS Command Line Interface を使用する例については、「Amazon CloudWatch Internet Monitor での CLI の使用例」を参照してください。

コンソールを使用してモニターを作成するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. 左側のナビゲーションペインで、[ネットワークモニタリング] の下にある [Internet Monitor] を選択します。
3. [Create monitor] (モニターの作成) を選択します。
4. [Monitor name] (モニタ名) に、Internet Monitor でこのモニタに使用する名前を入力します。
5. [Add resources] (リソースを追加) を選択し、このモニタに使用する Internet Monitor の監視境界の設定対象となるリソースを選択します。

> **Note**
>
> 以下の点に注意してください。
>
> - Internet Monitor で意味のある出力を生成するには、インターネットゲートウェイが設定された状態で、追加する VPC がインターネットに接続されている必要があります。
> - VPC と CloudFront ディストリビューションを組み合わせて追加することも、WorkSpaces ディレクトリを追加することも、Network Load Balancer を追加することもできます。Network Load Balancer や WorkSpaces ディレクトリを他の種類のリソースと一緒に追加することはできません。

6. モニタリングするインターネットトラフィックのパーセンテージを選択します。
7. 必要に応じて、[詳細設定] で追加のオプションを指定します。

   - [都市ネットワークの最大値] で、Internet Monitor がトラフィックをモニタリングする都市ネットワーク (ロケーションと ASN、またはインターネットサービスプロバイダー) の数の上限を選択します。モニタを編集すれば、いつでも最大数を変更できます。「都市ネットワークの上限を選択する」を参照してください。

     デフォルトにリセットするには、`500000` と入力します。

     都市ネットワークの上限を設定すると、モニタリング対象として選択したトラフィックの割合にかかわらず、Internet Monitor がアプリケーションについてモニタリングする都市ネットワークの数に上限が設定されます。
   - 必要に応じて、Amazon S3 バケット名とカスタムプレフィックスを指定して、モニタリング対象のすべての都市ネットワークのインターネット測定値を Amazon S3 に公開できます。

     Internet Monitor は、アプリケーションの (トラフィック量で) 上位 500 のインターネット測定値を 5 分ごとに CloudWatch Logs にパブリッシュします。測定値を S3 にパブリッシュすることを選択しても、測定値は引き続き CloudWatch Logs にパブリッシュされます。詳細については、「Amazon CloudWatch Internet Monitor で Amazon S3 にインターネット測定値をパブリッシュする」を参照してください。
   - 必要に応じて、モニタにタグを追加できます。
8. [Create monitor] (モニターの作成) を選択します。

モニターを作成した後は、いつでもモニターを編集して、例えば、アプリケーショントラフィックの割合を変更したり、都市ネットワークの上限を更新したりできるほか、リソースを追加または削除することもできます。モニターを削除することもできます。これらのタスクを実行するには、Internet Monitor コンソールでモニターを選択し、[アクション] メニューからオプションを選択します。モニターの名前は変更できないことに注意してください。

Internet Monitor ダッシュボードを表示するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで、[ネットワークモニタリング]、[Internet Monitor] の順に選択します。

   [Monitors] (モニタ) タブには、作成済みのモニタが一覧表示されます。

特定のモニターの詳細情報を表示するには、モニターを選択します。

リソースのモニターへの追加

モニターを作成するときは、アプリケーションのリソース (Amazon 仮想プライベートクラウド (VPC)、Network Load Balancer、Amazon CloudFront ディストリビューション、Network Load Balancers (NLB)、Amazon WorkSpaces ディレクトリ) のいずれかをそのモニターに関連付けます。次に、Internet Monitor は、アプリケーションのインターネット接続トラフィックとクライアントの場所を把握し、モニターに公開する関連測定値を判定するトラフィックプロファイルを作成し、保持します。

Internet Monitor のモニターには、「モニタリング対象リソース」として次のリソースを追加できます。Internet Monitor では、1 つのモニターに異なるタイプのリソースを追加することはできません。

- VPC: リージョンに追加する各 VPC はモニタリング対象リソースです。VPC を追加すると、Internet Monitor はその VPC 内のインターネット向けアプリケーションのトラフィックをすべてモニタリングします。例えば、Amazon EC2 インスタンスでホストされているアプリケーション、Network Load Balancer または Application Load Balancer の背後にあるアプリケーション、AWS Fargate コンテナなどです。
- 追加する各 Network Load Balancer は監視対象リソースです。
- CloudFront ディストリビューション: 追加する各 CloudFront ディストリビューションは、モニタリング対象リソースです。
- WorkSpaces ディレクトリ: リージョンで追加する各 WorkSpaces ディレクトリは、モニタリング対象リソースです。

VPC のトラフィックをモニタリングすると、VPC の背後にあるロードバランサーでホストされているアプリケーションのトラフィックが監視されます。複数のロードバランサーを備えた VPC を監視する代わりに、個々の Network Load Balancer ロードバランサーのトラフィックをモニタリングすることを選択できます。これは、例えば、ロードバランサーレベルでパフォーマンスや効率を向上させるための機能を理解して設定する必要がある場合に役立ちます。または、Network Load Balancer レベルのコンプライアンス情報が必要な場合があります。

Internet Monitor でリソースをモニターに追加する際には、次の点に注意してください。

- Internet Monitor で意味のある出力を生成するには、インターネットゲートウェイが設定された状態で、追加する VPC がインターネットに接続されている必要があります。
- Internet Monitor では、1 つのモニターに異なるタイプのリソースを追加することはできません。

VPC や NLB をリソースとして追加する場合、オプトインリージョンでは、リージョンごとに違いがあることに留意する必要があります。詳細については、「Amazon CloudWatch Internet Monitor がサポートされる AWS リージョン」を参照してください。

さらに、ラストマイルのレイテンシーの測定に関しては、リソースごとに違いがあります。Internet Monitor のレイテンシーの測定では、VPC、NLB および WorkSpaces ディレクトリにはラストマイルのレイテンシーは含まれません。

モニタリングするアプリケーショントラフィックの割合の選択

モニタリングするアプリケーショントラフィックの割合として選択したカバレッジによって、アプリケーションの都市ネットワーク (クライアントのロケーションと ASN、通常はインターネットサービスプロバイダー) の数が決まります。この数の上限は、設定可能なオプションの都市ネットワークの上限です。

アプリケーショントラフィックの 100% 未満をモニタリングすることを選択した場合、モニターにはオブザーバビリティのギャップが存在する可能性があります。これは、トラフィックをモニタリングしていない場合に Amazon CloudWatch Internet Monitor が作成するヘルスイベントがある場合、それらの問題に気付くことができないためです。また、アプリケーションに対するクライアントのアクセスに関するパフォーマンスとアベイラビリティのスコア情報のカバレッジが狭まる可能性があります。

次のセクションでは、トラフィック割合の設定とカバレッジを調べたり、カバレッジの増減による影響を把握したりするためのオプションについて説明します。

- アプリケーションのトラフィックの割合の変更を検討する

- さまざまなトラフィックの割合の設定でモニタリングされている都市ネットワークの数を表示する

アプリケーションのトラフィックの割合の変更を検討する

割合を変更する際にモニタリングされる都市ネットワークの数を確認することで、アプリケーショントラフィックの割合を変更する値を検討できます。このセクションの手順では、ステップバイステップの情報を提供します。

Internet Monitor コンソールでは、モニターのアプリケーショントラフィックの割合の増減を試みて、その結果として対象となる都市ネットワークの推定数を表示できます。このオプションを使用すると、トラフィックの割合を変更することで、モニタリングされる都市モニターの数にどのような影響が生じるのかをすぐに確認できます。これは、アプリケーションのために選択すべき適切なアプリケーショントラフィックの割合を把握するのに役立ちます。

アプリケーショントラフィックの割合を増減してモニタリングの範囲を検討するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. 左側のナビゲーションペインで、[ネットワークモニタリング] の下にある [Internet Monitor] を選択します。
3. モニターのリストからモニターを選択します。
4. [概要] タブの [モニタリング対象トラフィック] セクションで、割合のグラフを選択し、[モニタリングカバレッジの更新] を選択します。
5. [トラフィックのモニタリングカバレッジの確認と設定] ダイアログで、矢印をクリックしてモニタリングするトラフィックの割合を増減します。トラフィックの 100% を選択すると、アプリケーションのモニタリングのために、フルカバレッジの対象となっているモニタリング対象の都市ネットワークの数がわかります。
6. モニタリング対象の都市ネットワークの数 (ここで推定されます) がコストにどのように影響するのかの詳細については、CloudWatch 料金見積りツールへのリンクをクリックして、Internet Monitor まで下方向にスクロールします。
7. モニタリングするトラフィックの新しい割合を設定するには、[モニターカバレッジの更新] を選択します。または、現在のカバレッジレベルを維持するには、[キャンセル] を選択します。

さまざまなトラフィックの割合の設定でモニタリングされている都市ネットワークの数を表示する

アプリケーションについてモニタリングされる都市ネットワークの数を、さまざまなアプリケーショントラフィックの割合で表示できます。このセクションの手順では、ステップバイステップの情報を提供します。

Internet Monitor コンソールでは、指定した時間間隔にわたって、さまざまなアプリケーショントラフィックの割合で、都市ネットワークのカバレッジがどのように変化するかを示すグラフを表示できます。これにより、特定のトラフィックの割合におけるアプリケーションのモニタリングカバレッジをすべて 1 つのグラフで迅速に視覚化して比較できます。

アプリケーショントラフィックの割合と、対応する都市ネットワークのカバレッジのグラフを表示するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. 左側のナビゲーションペインで、[ネットワークモニタリング] の下にある [Internet Monitor] を選択します。
3. モニターのリストからモニターを選択します。
4. [トラフィックインサイト] タブを選択し、[インターネットトラフィックのグラフ] まで下方向にスクロールします。
5. [トラフィックカバレッジのオプションを比較] のドロップダウンリストで、1 つ以上の割合を選択します。1 つ以上のアプリケーショントラフィックの割合を選択すると、[モニタリング対象都市ネットワークの総数] のグラフが更新され、Internet Monitor がそのトラフィックの割合について提供するモニタリングカバレッジが表示されます。[都市ネットワーク (トラフィックの 100%)] を選択すると、アプリケーションのモニタリングのために、フルカバレッジの対象となっているモニタリング対象の都市ネットワークの数がわかります。

以下に留意してください。

- トラフィックカバレッジは、過去 1 時間のアプリケーショントラフィックにおける、都市ネットワークの数に基づいて計算されます。つまり、モニタリングするトラフィックの割合を指定した後は、このトラフィックカバレッジを比較したグラフに表示されている数よりも、アプリケーションでモニタリングされる都市ネットワークの数が少なくなる可能性があります。
- すべてのアプリケーショントラフィックが確実にモニタリングされるようにするには、`TrafficPercentageToMonitor` を 100 に設定し、`MaxCityNetworksToMonitor` を設定しないでください。または、`MaxCityNetworksToMonitor` をインターネットモニターの上限である 500,000 に設定することもできます。
- 都市ネットワークの上限を設定している場合、選択したアプリケーショントラフィックの割合のオプションにかかわらず、モニタリング対象の都市ネットワークの総数が、その上限を超えることはありません。

- モニタリング対象の都市ネットワークの数がコストに及ぼす影響の詳細を知ることができます。CloudWatch の料金見積りツールのページで、Internet Monitor まで下方向にスクロールします。

モニタリングするトラフィックの新しい割合を設定するには、[他のトラフィックカバレッジオプションを確認] で、[モニタリングカバレッジの更新] を選択します。ダイアログで、トラフィックの割合を選択し、[モニターカバレッジの更新] を選択します。

都市ネットワークの上限を選択する

Amazon CloudWatch Internet Monitor は、クライアントがアプリケーションリソースにアクセスする一部または全部の口ケーション、およびそのアクセスのために経由するすべての ASN (通常はインターネットサービスプロバイダー) のアプリケーショントラフィック、すなわち、お客様のアプリケーションインターネットトラフィックの都市ネットワークをモニタリングできます。モニターを作成する際、モニタリングするアプリケーショントラフィックの割合を選択します。これは、モニターを編集することで、いつでも更新できます。

トラフィックの割合を設定することに加えて、モニタリングする都市ネットワークの数の上限も設定できます。このセクションでは、都市ネットワークの制限が請求コストの管理にどのように役立つかを説明し、設定する制限を決定するのに役立つ情報と例を提供します。

都市ネットワークの上限を設定すると、請求額を予測しやすくなります。詳細については、「Amazon CloudWatch 料金表」をご覧ください。また、CloudWatch 料金見積りツールを使用すると、実際にモニタリング対象となる都市ネットワークの数のさまざまな値が請求額にどのように影響を与えるかを調べることができます。オプションを確認するには、CloudWatch の料金見積りツールのページで「Internet Monitor」まで下へスクロールします。

モニタを更新して都市ネットワークの最大数を変更するには、「Internet Monitor のモニターの編集または削除」を参照してください。

都市ネットワークの上限に応じた課金の仕組み

モニタリングする都市ネットワークの数に上限を設けることで、想定外のコストの請求を回避するのに役立ちます。これは、例えばトラフィックパターンが広範に及ぶ場合に有益です。請求額は、(アカウントごとのモニター全体で) 最初の 100 個の都市ネットワークを超えると、モニタリングする都市ネットワークごとに増加します。都市ネットワークの上限を設定すると、モニタリング対象として選択したトラフィックの割合にかかわらず、Internet Monitor がアプリケーションについてモニタリングする都市ネットワークの数に上限が設定されます。

実際にモニタリングした都市ネットワークの数に対してのみ課金されます。選択した都市ネットワークの最大数により、Internet Monitor がモニタでトラフィックをモニタリングする都市ネットワークの合計に上限を設定できます。モニタを編集すれば、いつでも最大数を変更できます。

オプションを確認するには、CloudWatch の料金見積りツールのページで「Internet Monitor」まで下へスクロールします。Internet Monitor の料金の詳細については、「Amazon CloudWatch 料金表」のページの「Internet Monitor」セクションを参照してください。

都市ネットワークの上限を選択する方法

選択する都市ネットワークの上限を決定する際には、アプリケーションでモニタリングするトラフィック量を考慮することをお勧めします。Internet Monitor の `CityNetworksMonitored`、`TrafficMonitoredPercent`、1 つ以上の `CityNetworksFor`*`NN`*`PercentTraffic` メトリクスは、モニタを作成した後、トラフィックの使用量とカバレッジを分析するのに役立ちます。*NN* はパーセンテージの値であり、25、50、90、95、99、100 のいずれかになります これらのメトリクスや他の Internet Monitor のメトリクスすべての定義を確認するには、「Amazon CloudWatch Internet Monitor での CloudWatch Metrics の使用」を参照してください。

インターネットトラフィックカバレッジの概要グラフを表示するには、CloudWatch ダッシュボードで [トラフィックインサイト] タブに移動し、[インターネットトラフィックのグラフ] セクションで、[トラフィックカバレッジのオプションを比較] のオプションを選択します。このセクションで表示されるグラフには、アプリケーションについてモニタリングされている都市ネットワークの実際の数と、ドロップダウンリストで選択したさまざまなアプリケーショントラフィックの割合のグラフの線が表示されます。詳細については、「アプリケーショントラフィックの割合の設定」を参照してください。

オプションの詳細を確認するには、次に説明する例のように、Internet Monitor のメトリクスを使用できます。これらの例は、アプリケーションのインターネットトラフィックカバレッジの希望範囲に応じて、どのように都市ネットワークの最大数を最適に選択するかを示しています。CloudWatch メトリクスの Internet Monitor メトリクスのクエリを使用すると、アプリケーションのインターネットトラフィックカバレッジの詳細を理解するのに役立ちます。

都市ネットワークの上限の値の決定例

例として、モニタリング対象の上限を 100 の都市ネットワークに設定し、2,637 の都市ネットワークのクライアントがアプリケーションにアクセスしているとします。CloudWatch メトリクスでは、次の Internet Monitor メトリクスが返されます。

```
CityNetworksMonitored 100
```

```
TrafficMonitoredPercent  12.5
CityNetworksFor90PercentTraffic  2143
CityNetworksFor100PercentTraffic  2637
```

この例からは、その時点でモニタリングしているのはインターネットトラフィックの 12.5% であり、最大数は 100 の都市ネットワークに設定されていることがわかります。トラフィックの 90% をモニタリングしたい場合、次のメトリクスでその情報が得られます。CityNetworksFor90PercentTraffic は、90% のカバレッジを得るには、2,143 の都市ネットワークをモニタリングする必要があることを示します。そのためにはモニタを更新し、都市ネットワークの最大数を 2,143 に設定します。

同様に、アプリケーションのインターネットトラフィックの 100% をモニタリングしたいとします。次のメトリクス CityNetworksFor100PercentTraffic は、そのためにモニタを更新して、都市ネットワークの最大数を 2,637 に設定する必要があることを示します。

都市ネットワークの最大数を 5,000 に設定した場合、その数は 2,637 を超えているため、次のメトリクスが返されます。

```
CityNetworksMonitored 2637
TrafficMonitoredPercent  100
CityNetworksFor90PercentTraffic  2143
CityNetworksFor100PercentTraffic  2637
```

これらのメトリクスから、高めの上限にすると、インターネットトラフィックの 100% である 2,637 の都市ネットワークすべてをモニタリングすることがわかります。

Amazon CloudWatch Internet Monitor で Amazon S3 にインターネット測定値をパブリッシュする

Amazon CloudWatch Internet Monitor が Amazon S3 に、モニタ内のモニタリング対象の都市ネットワークに対するインターネット向けトラフィックのインターネット測定値をパブリッシュすることを選択できます (都市ネットワークとは、クライアントロケーションと ASN、通常はインターネットサービスプロバイダーのことです)。ただし、都市ネットワークのサービス上限は 500,000 です。Internet Monitor は、各モニタの (トラフィック量で) 上位 500 の都市ネットワークのインターネット測定値を、5 分ごとに CloudWatch Logs に自動的にパブリッシュします。S3 にパブリッシュされる測定値には、CloudWatch Logs にパブリッシュされている上位 500 が含まれます。

モニターを作成または更新する際、S3 に発行するオプションを選択し、測定値を発行するバケットを指定できます。バケットを S3 で先に作成してから、Internet Monitor で指定する必要がありま

す。S3 にパブリッシュするインターネット測定値の都市ネットワークのサービス上限は 500,000 です。Internet Monitor は、インターネット測定値をイベント (バケットに保存された一連の圧縮ログファイルオブジェクト) として S3 にパブリッシュします。

Internet Monitor が測定値をパブリッシュするための S3 バケットを作成する際、CloudWatch Logs が提供するアクセス許可のガイダンスに必ず従ってください。これにより、Internet Monitor はログを S3 に直接パブリッシュできます。また AWS は必要に応じて、そのログを受け取るロググループに関連付いたリソースポリシーの作成や変更ができます。詳細については、「Amazon CloudWatch Logs ユーザーガイド」の「CloudWatch Logs に送信されたログ」を参照してください。

パブリッシュするログファイルは圧縮されます。Amazon S3 コンソールを使用してログファイルを開くと、ファイルは解凍されてインターネット測定値のイベントが表示されます。ファイルをダウンロードする場合は、イベントを表示するのに解凍する必要があります。

Amazon Athena を使用し、ログファイルのインターネット測定値に対してクエリを実行することもできます。Amazon Athena はインタラクティブなクエリサービスであり、Amazon S3 内のデータを標準 SQL を使用して簡単に分析できます。詳細については、「Amazon Athena を使用して、Amazon S3 ログファイルのインターネット測定値をクエリする」を参照してください。

Internet Monitor のモニタを使用する

Amazon CloudWatch Internet Monitor のモニタを作成すると、さまざまな使い道があります。例えば、CloudWatch ダッシュボードで情報を表示したり、AWS Command Line Interface を使用して情報を取得したり、ヘルスアラートを設定したりできます。

モニターはアプリケーションと構成設定に関する情報を提供するため、Internet Monitor は測定値とメトリクスをカスタマイズしてイベントに発行できます。Internet Monitor は、AWS のグローバルインフラストラクチャフットプリントから測定値を収集します。これらの測定値は、世界中の膨大な量のネットワークパフォーマンスとアベイラビリティの情報です。Internet Monitor は、アプリケーションに追加したリソースからの情報を使用して、アプリケーションがアクティブな都市ネットワークを対象とするパフォーマンスと可用性の測定値をパブリッシュします (都市ネットワークとは、クライアントロケーションと ASN、通常はインターネットサービスプロバイダー (ISP) のことです)。そのため、Internet Monitor ダッシュボードと CloudWatch Logs の測定値やメトリクス (可用性、パフォーマンス、モニタリング対象転送バイト数、往復時間など) は、クライアントロケーションと ASN によって異なります。

Internet Monitor は、パフォーマンスと可用性に異常があるかどうかも判断します。Internet Monitor はデフォルトで、クライアントロケーション内の送信元と宛先のペアごとに AWS が収集した可用性とパフォーマンスに関する測定値をトラフィックに重ね合わせます。これにより、パフォーマンスま

たは可用性に顕著な低下が生じた時期を判断します。アプリケーションのロケーションや対象範囲で大幅な劣化が発生すると、Internet Monitor はヘルスイベントを生成し、問題に関する情報をモニタにパブリッシュします。

モニターを作成すると、そのモニターを使用して、Internet Monitor が提供する情報に以下の方法でアクセスしたりアラートを受け取ったりできます。

- CloudWatch ダッシュボードを使用して、パフォーマンス、可用性、ヘルスイベントを表示して調べ、アプリケーションの履歴データを確認し、パフォーマンスを向上させるべく、アプリケーションの新しい設定方法を見出すために役立つインサイトを取得することができます。詳細については、以下をご覧ください。
  - Amazon CloudWatch Internet Monitor ([Overview] (概要) タブ) によるパフォーマンスと可用性のリアルタイムでの追跡
  - Amazon CloudWatch Internet Monitor ([Historical Explorer] タブ) でデータをフィルタリングおよび表示する
  - Amazon CloudWatch Internet Monitor (トラフィックインサイトのタブ) でアプリケーションのパフォーマンスを改善するためのインサイトを取得する
- ヘルスイベントのしきい値を設定して、Internet Monitor がアプリケーションのヘルスイベントを作成するトリガーを変更してください。全体のしきい値とローカル (都市ネットワーク) のしきい値を設定できます。詳細については、「ヘルスイベントのしきい値の変更」を参照してください。
- Internet Monitor API アクションで AWS CLI コマンド を使用して、トラフィックプロファイル情報の表示、測定値の表示、ヘルスイベントの一覧表示などが行えます。詳細については、「Amazon CloudWatch Internet Monitor での CLI の使用例」を参照してください。
- CloudWatch Contributor Insights、CloudWatch メトリクスエクスプローラー、CloudWatch Logs Insights などの標準の CloudWatch ツールを使用して、CloudWatch のデータを可視化します。詳細については、「CloudWatch ツールと Internet Monitor のクエリインターフェイスによるデータの調査」を参照してください。
- S3 への測定値のパブリッシュを有効にしている場合、Athena と S3 ログを使用して、アプリケーションの Internet Monitor インターネット測定値にアクセスして分析することができます。
- Amazon EventBridge 通知を作成して、Internet Monitor がヘルスイベントの発生を把握したときにアラートを受け取ります。詳細については、「Amazon EventBridge での Amazon CloudWatch Internet Monitor の使用」を参照してください。
- 問題の原因が AWS のネットワークにあると Internet Monitor が判断した場合、AWS Health Dashboard 通知を自動的に受け取ります。通知には、問題を軽減するために AWS で行われている手順が含まれます。

Internet Monitor のモニターの編集または削除

Amazon CloudWatch Internet Monitor でモニターを作成した後に、[アクション] メニューを使用すると、そのモニターを編集または削除できます。例えば、モニターを編集して次を実行できます。

- モニタリングするアプリケーショントラフィックの割合を変更する
- 都市ネットワークの上限を設定または更新する
- アベイラビリティまたはパフォーマンスのスコアのヘルスイベントのしきい値を変更する
- リソースを追加または削除する
- Amazon S3 に対するイベントの発行を有効化または更新する

モニタを削除することもできます。モニターの作成後はモニターの名前を変更できないことに注意してください。

モニタを変更する、またはモニタを削除するには、以下のいずれかの手順に従ってください。

モニタを編集する

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. 左側のナビゲーションペインで、[ネットワークモニタリング] の下にある [Internet Monitor] を選択します。
3. モニタを選択し、[アクション] メニューを選択します。
4. [モニタの更新] を選択します。
5. 必要な更新を行います。例えば、モニタリングするトラフィックの割合を変更するには、[モニタリングするアプリケーショントラフィック] で割合を選択または入力します。
6. [Update] (更新) を選択します。

モニタを削除する

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. 左側のナビゲーションペインで、[ネットワークモニタリング] の下にある [Internet Monitor] を選択します。
3. モニタを選択し、[アクション] メニューを選択します。
4. [無効化] を選択します。
5. [アクション] メニューを再度選択してから [削除] を選択します。

更新できるオプションの詳細については、次を参照してください。

- Internet Monitor に追加するリソースの詳細については、「リソースのモニターへの追加」を参照してください。
- アプリケーショントラフィックの割合の詳細については、「モニタリングするアプリケーショントラフィックの割合の選択」を参照してください。
- ヘルスイベントのしきい値の変更の詳細については、「ヘルスイベントのしきい値を変更する」を参照してください。
- 都市ネットワークの上限の詳細については、「都市ネットワークの上限を選択する」を参照してください。
- S3 に対するイベントの発行の選択の詳細については、「Amazon CloudWatch Internet Monitor で Amazon S3 にインターネット測定値をパブリッシュする」を参照してください。

Amazon VPC で Amazon CloudWatch Internet Monitor を追加または作成する

AWS Management Console を使用して Amazon 仮想プライベートクラウド (VPC) を作成する場合、Amazon CloudWatch Internet Monitorでのモニタリングをオプションで設定することもできます。既存のモニターに VPC を追加することも、VPC 用の新しいモニターを Amazon VPC コンソールで作成することもできます。

VPC で Internet Monitor を使用することで、特定のアプリケーションのクライアントロケーションと ASN (通常はインターネットサービスプロバイダー) について、可用性やパフォーマンス、モニタリング対象の転送バイト数、往復時間に関する測定値とメトリクスを表示し、評価できます。Internet Monitor は、いつパフォーマンスと可用性に異常が起こるのかを判断し、モニターにヘルスイベントを作成します。このイベントについての通知を受けるかどうかは、選択が可能です。モニターを使用して顧客のアプリケーション体験を管理し、改善する方法についての詳細は、「Internet Monitor のモニタを使用する」を参照してください。

> **Important**
>
> モニターを作成したり、既存のモニターに VPC を追加したりするには、適切なアクセス許可が必要です。詳細については、「Amazon CloudWatch Internet Monitor 用 Identity and Access Management」を参照してください。

既存のモニターに VPC を追加する

AWS Management Console で VPC を作成するときに、Amazon CloudWatch Internet Monitor が、既存のモニターに新しい VPC を追加するように選択できます。VPC を追加してから数分待つと、VPC のメトリクスが Internet Monitor コンソールに表示されます。

モニターを編集すれば、いつでもその VPC を削除したり、別の VPC や他のリソースを追加したりできます。モニタリングするトラフィックの割合を変更したり、その他の変更を行うこともできます。VPC をモニターから削除するよう選択した場合、クライアントからその VPC へのトラフィックは、Internet Monitor によってモニタリングされなくなります。

モニターの更新について詳しくは、「Internet Monitor のモニターの編集または削除」を参照してください。

VPC のモニターを作成する

VPC のモニターを作成する場合は、[モニターの作成] ウィザードの指示に従って手順を進めます。モニターを作成する際に、VPC をモニタリング対象リソースとして追加します。必要に応じて、アプリケーションでモニタリングするクライアントトラフィックの割合を選択することもできます (デフォルトでは 100%)。

詳細については、「コンソールを使用して Amazon CloudWatch Internet Monitor でモニターを作成する」に記載されている情報を確認してください。

料金

Amazon CloudWatch Internet Monitor は従量課金制です。Internet Monitor には、モニタリング対象リソースごとと都市ネットワークごとの 2 つの料金体系があります。都市ネットワークとは、クライアントがアプリケーションリソースにアクセスするロケーションと、リソースにアクセスする際に経由するネットワーク (インターネットサービスプロバイダー (ISP) などの ASN) のことです。

料金例などの詳細については、「Amazon CloudWatch Internet Monitor の料金」を参照してください。

VPC のモニタリングを停止する

Internet Monitor による VPC リソースのモニタリングを停止する場合は、Internet Monitor コンソールで次の手順を実行します。

モニターからリソースを削除するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。

2. 左側のナビゲーションペインで、[ネットワークモニタリング] の下にある [Internet Monitor] を選択します。
3. モニタを選択し、[アクション] メニューを選択します。
4. [モニタの更新] を選択します。
5. [追加リソース] で [リソースを削除] を選択します。
6. 削除する VPC を選択してから、[削除]を選択します。
7. [Update] (更新) を選択します。

CloudFront を使用して、Amazon CloudWatch Internet Monitor のモニターを追加または作成する

Amazon CloudFront コンソールでのディストリビューション用のメトリクスダッシュボードでは、Amazon CloudWatch Internet Monitor のディストリビューションに追加のモニタリングを設定できます。ディストリビューションを既存のモニターに追加するか、ディストリビューション用に新しいモニターを作成します。

CloudFront ディストリビューションで Internet Monitor を使用することで、特定のアプリケーションのクライアントロケーションと ASN (通常はインターネットサービスプロバイダー) について、可用性やパフォーマンス、モニタリング対象の転送バイト数、往復時間に関する測定値とメトリクスを表示し、評価できます。Internet Monitor は、いつパフォーマンスと可用性に異常が起こるのかを判断し、モニターにヘルスイベントを作成します。このイベントについての通知を受けるかどうかは、選択が可能です。モニターを使用して顧客のアプリケーション体験を管理し、改善する方法についての詳細は、「Internet Monitor のモニタを使用する」を参照してください。

> **Important**
>
> モニターを作成したり、既存のモニターにディストリビューションを追加したりするには、適切なアクセス許可が必要です。詳細については、「Amazon CloudWatch Internet Monitor 用 Identity and Access Management」を参照してください。

既存のモニターにディストリビューションを追加する

Internet Monitor が、AWS Management Consoleの CloudFront メトリクスダッシュボードから既存のモニターにディストリビューションを直接追加するように設定できます。ディストリビューションを追加してから数分待つと、Internet Monitor コンソールにディストリビューションのメトリクスが表示されます。

モニターを編集することで、いつでもディストリビューションを削除したり、別のディストリビューションや他のリソースを追加したりできます。モニタリングするトラフィックの割合を変更したり、その他の変更を行うこともできます。モニターからディストリビューションを削除することを選択すると、Internet Monitor はクライアントからそのディストリビューションへのトラフィックのモニタリングを停止します。

モニターの更新について詳しくは、「Internet Monitor のモニターの編集または削除」を参照してください。

ディストリビューション用のモニターを作成する

ディストリビューション用のモニターを作成する場合は、[モニターを作成] ウィザードの指示に従って手順を実行します。モニターを作成する際は、ディストリビューションをモニタリング対象リソースとして追加します。必要に応じて、アプリケーションでモニタリングするクライアントトラフィックの割合を選択することもできます (デフォルトでは 100%)。

詳細については、「コンソールを使用して Amazon CloudWatch Internet Monitor でモニターを作成する」に記載されている情報を確認してください。

料金

Amazon CloudWatch Internet Monitor は従量課金制です。Internet Monitor には、モニタリング対象リソースごとと都市ネットワークごとの 2 つの料金体系があります。都市ネットワークとは、クライアントがアプリケーションリソースにアクセスするロケーションと、リソースにアクセスする際に経由するネットワーク (インターネットサービスプロバイダー (ISP) などの ASN) のことです。

料金例などの詳細については、「Amazon CloudWatch Internet Monitor の料金」を参照してください。

ディストリビューションのモニタリングを停止する

Internet Monitor によるディストリビューションリソースのモニタリングを停止する場合は、Internet Monitor コンソールで次の手順を実行します。

モニターからリソースを削除するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. 左側のナビゲーションペインで、[ネットワークモニタリング] の下にある [Internet Monitor] を選択します。
3. モニタを選択し、[アクション] メニューを選択します。

4. [モニタの更新] を選択します。
5. [追加リソース] で [リソースを削除] を選択します。
6. 削除するディストリビューションを選択してから、[削除]を選択します。
7. [Update] (更新) を選択します。

Amazon CloudWatch Internet Monitor での CLI の使用例

このセクションでは、Amazon CloudWatch Internet Monitor のオペレーションで、AWS Command Line Interface を使用する例を示します。

開始する前に、モニタリングする Amazon 仮想プライベートクラウド (VPC)、Network Load Balancers、Amazon CloudFront ディストリビューション、または Amazon WorkSpaces ディレクトリがある AWS アカウントと同じアカウントで AWS CLI を使用するようにログインする必要があります。Internet Monitor はアカウントをまたいだリソースへのアクセスをサポートしていません。AWS CLI の使用の詳細については、「AWS CLI コマンドリファレンス」を参照してください。Amazon CloudWatch Internet Monitorで API アクションを使用する方法の詳細については、「Amazon CloudWatch Internet Monitor API Reference Guide」(Amazon CloudWatch Internet Monitor API リファレンスガイド) を参照してください。

トピック

モニターを作成する

Internet Monitor でモニタを作成する際は、アプリケーションのインターネットトラフィックがある場所を表示するために、モニタに名前を付けてリソースを関連付けます。監視するアプリケーショントラフィックの量を定義するトラフィックの割合を指定します。これにより、監視対象となる都市ネットワーク、つまりクライアントの場所やASN (通常はインターネットサービスプロバイダーやISP) の数も決まります。また、アプリケーションリソースをモニタリングする都市ネットワークの

最大数に上限を設定して、請求額を抑えることができます。詳細については、「都市ネットワークの上限を選択する」を参照してください。

さらに、アプリケーションのすべてのインターネット測定値を、Amazon S3 にパブリッシュするかどうかを選択できます。上位 500 位 の都市ネットワーク (トラフィック量で判断) のインターネット測定値は、Internet Monitor によって CloudWatch Logs に自動的に公開されますが、すべての測定値を S3 に公開することも選択できます。

AWS CLI を使用してモニタを作成するには、`create-monitor` コマンドを使用します。次のコマンドでは、トラフィックを 100% 監視するモニターを作成していますが、都市ネットワークの最大制限は 10,000 に設定されており、VPC リソースを追加して、インターネット測定値は Amazon S3 にパブリッシュする選択になっています。

> **Note**
>
> Internet Monitor は、インターネット測定値を 5 分ごとに CloudWatch Logs にパブリッシュします。対象は、各モニタにトラフィックを送信する上位 500 の都市ネットワーク (クライアントロケーションと ASN、通常はインターネットサービスプロバイダー (ISP)) です。必要に応じて、モニタリング対象の都市ネットワークすべてのインターネット測定値を Amazon S3 バケットにパブリッシュする選択ができます (ただし、サービスの上限数は 500,000 の都市ネットワークです)。詳細については、「Amazon CloudWatch Internet Monitor で Amazon S3 にインターネット測定値をパブリッシュする」を参照してください。

```
aws internetmonitor --create-monitor monitor-name "TestMonitor" \
    --traffic-percentage-to-monitor 100 \
    --max-city-networks-to-monitor 10000 \
    --resources "arn:aws:ec2:us-east-1:111122223333:vpc/vpc-11223344556677889" \
    --internet-measurements-log-delivery
 S3Config="{BucketName=MyS3Bucket,LogDeliveryStatus=ENABLED}"
```

```
{
"Arn": "arn:aws:internetmonitor:us-east-1:111122223333:monitor/TestMonitor",
 "Status": "ACTIVE"
}
```

> **Note**
>
> モニタの名前は変更できないことに注意してください。

モニターの詳細の表示

AWS CLI を使用してモニタに関する情報を表示するには、get-monitor コマンドを使用します。

```
aws internetmonitor get-monitor --monitor-name "TestMonitor"
```

```
{
    "ClientLocationType": "city",
    "CreatedAt": "2022-09-22T19:27:47Z",
    "ModifiedAt": "2022-09-22T19:28:30Z",
    "MonitorArn": "arn:aws:internetmonitor:us-east-1:111122223333:monitor/TestMonitor",
    "MonitorName": "TestMonitor",
    "ProcessingStatus": "OK",
    "ProcessingStatusInfo": "The monitor is actively processing data",
    "Resources": [
        "arn:aws:ec2:us-east-1:111122223333:vpc/vpc-11223344556677889"
    ],
    "MaxCityNetworksToMonitor": 10000,
    "Status": "ACTIVE"
}
```

ヘルスイベントを一覧表示する

アプリケーションのインターネットトラフィックについてのパフォーマンスが低下した場合、Internet Monitor はモニタでヘルスイベントを作成します。AWS CLI を使用して現在のヘルスイベントの一覧を表示するには、list-health-events コマンドを使用します。

```
aws internetmonitor list-health-events --monitor-name "TestMonitor"
```

```
{
    "HealthEvents": [
        {
            "EventId": "2022-06-20T01-05-05Z/latency",
            "Status": "RESOLVED",
            "EndedAt": "2022-06-20T01:15:14Z",
```

```
            "ServiceLocations": [
                {
                    "Name": "us-east-1"
                }
            ],
            "PercentOfTotalTrafficImpacted": 1.21,
            "ClientLocations": [
                {
                    "City": "Lockport",
                    "PercentOfClientLocationImpacted": 60.370000000000005,
                    "PercentOfTotalTraffic": 2.01,
                    "Country": "United States",
                    "Longitude": -78.6913,
                    "AutonomousSystemNumber": 26101,
                    "Latitude": 43.1721,
                    "Subdivision": "New York",
                    "NetworkName": "YAHOO-BF1"
                }
            ],
            "StartedAt": "2022-06-20T01:05:05Z",
            "ImpactType": "PERFORMANCE",
            "EventArn": "arn:aws:internetmonitor:us-east-1:111122223333:monitor/
TestMonitor/health-event/2022-06-20T01-05-05Z/latency"
        },
        {
            "EventId": "2022-06-20T01-17-56Z/latency",
            "Status": "RESOLVED",
            "EndedAt": "2022-06-20T01:30:23Z",
            "ServiceLocations": [
                {
                    "Name": "us-east-1"
                }
            ],
            "PercentOfTotalTrafficImpacted": 1.29,
            "ClientLocations": [
                {
                    "City": "Toronto",
                    "PercentOfClientLocationImpacted": 75.32,
                    "PercentOfTotalTraffic": 1.05,
                    "Country": "Canada",
                    "Longitude": -79.3623,
                    "AutonomousSystemNumber": 14061,
                    "Latitude": 43.6547,
                    "Subdivision": "Ontario",
```

```
                    "CausedBy": {
                        "Status": "ACTIVE",
                        "Networks": [
                            {
                                "AutonomousSystemNumber": 16509,
                                "NetworkName": "Amazon.com"
                            }
                        ],
                        "NetworkEventType": "AWS"
                    },
                    "NetworkName": "DIGITALOCEAN-ASN"
                },
                {
                    "City": "Lockport",
                    "PercentOfClientLocationImpacted": 22.91,
                    "PercentOfTotalTraffic": 2.01,
                    "Country": "United States",
                    "Longitude": -78.6913,
                    "AutonomousSystemNumber": 26101,
                    "Latitude": 43.1721,
                    "Subdivision": "New York",
                    "NetworkName": "YAHOO-BF1"
                },
                {
                    "City": "Hangzhou",
                    "PercentOfClientLocationImpacted": 2.88,
                    "PercentOfTotalTraffic": 0.7799999999999999,
                    "Country": "China",
                    "Longitude": 120.1612,
                    "AutonomousSystemNumber": 37963,
                    "Latitude": 30.2994,
                    "Subdivision": "Zhejiang",
                    "NetworkName": "Hangzhou Alibaba Advertising Co.,Ltd."
                }
            ],
            "StartedAt": "2022-06-20T01:17:56Z",
            "ImpactType": "PERFORMANCE",
            "EventArn": "arn:aws:internetmonitor:us-east-1:111122223333:monitor/
TestMonitor/health-event/2022-06-20T01-17-56Z/latency"
        },
        {
            "EventId": "2022-06-20T01-34-20Z/latency",
            "Status": "RESOLVED",
            "EndedAt": "2022-06-20T01:35:04Z",
```

```
            "ServiceLocations": [
                {
                    "Name": "us-east-1"
                }
            ],
            "PercentOfTotalTrafficImpacted": 1.15,
            "ClientLocations": [
                {
                    "City": "Lockport",
                    "PercentOfClientLocationImpacted": 39.45,
                    "PercentOfTotalTraffic": 2.01,
                    "Country": "United States",
                    "Longitude": -78.6913,
                    "AutonomousSystemNumber": 26101,
                    "Latitude": 43.1721,
                    "Subdivision": "New York",
                    "NetworkName": "YAHOO-BF1"
                },
                {
                    "City": "Toronto",
                    "PercentOfClientLocationImpacted": 29.770000000000003,
                    "PercentOfTotalTraffic": 1.05,
                    "Country": "Canada",
                    "Longitude": -79.3623,
                    "AutonomousSystemNumber": 14061,
                    "Latitude": 43.6547,
                    "Subdivision": "Ontario",
                    "CausedBy": {
                        "Status": "ACTIVE",
                        "Networks": [
                            {
                                "AutonomousSystemNumber": 16509,
                                "NetworkName": "Amazon.com"
                            }
                        ],
                        "NetworkEventType": "AWS"
                    },
                    "NetworkName": "DIGITALOCEAN-ASN"
                },
                {
                    "City": "Hangzhou",
                    "PercentOfClientLocationImpacted": 2.88,
                    "PercentOfTotalTraffic": 0.7799999999999999,
                    "Country": "China",
```

```
                    "Longitude": 120.1612,
                    "AutonomousSystemNumber": 37963,
                    "Latitude": 30.2994,
                    "Subdivision": "Zhejiang",
                    "NetworkName": "Hangzhou Alibaba Advertising Co.,Ltd."
                }
            ],
            "StartedAt": "2022-06-20T01:34:20Z",
            "ImpactType": "PERFORMANCE",
            "EventArn": "arn:aws:internetmonitor:us-east-1:111122223333:monitor/
TestMonitor/health-event/2022-06-20T01-34-20Z/latency"
        }
    ]
}
```

特定のヘルスイベントを表示する

CLI を使用して特定のヘルスイベントに関する詳細な情報を表示するには、get-health-event コマンドにモニタ名とヘルスイベント ID を指定して実行します。

```
aws internetmonitor get-monitor --monitor-name "TestMonitor" --event-id "health-event/
TestMonitor/2021-06-03T01:02:03Z/latency"
```

```
{
    "EventId": "2022-06-20T01-34-20Z/latency",
    "Status": "RESOLVED",
    "EndedAt": "2022-06-20T01:35:04Z",
    "ServiceLocations": [
        {
            "Name": "us-east-1"
        }
    ],
    "EventArn": "arn:aws:internetmonitor:us-east-1:111122223333:monitor/TestMonitor/
health-event/2022-06-20T01-34-20Z/latency",
    "LastUpdatedAt": "2022-06-20T01:35:04Z",
    "ClientLocations": [
        {
            "City": "Lockport",
            "PercentOfClientLocationImpacted": 39.45,
            "PercentOfTotalTraffic": 2.01,
            "Country": "United States",
            "Longitude": -78.6913,
            "AutonomousSystemNumber": 26101,
```

```
            "Latitude": 43.1721,
            "Subdivision": "New York",
            "NetworkName": "YAHOO-BF1"
        },
        {
            "City": "Toronto",
            "PercentOfClientLocationImpacted": 29.770000000000003,
            "PercentOfTotalTraffic": 1.05,
            "Country": "Canada",
            "Longitude": -79.3623,
            "AutonomousSystemNumber": 14061,
            "Latitude": 43.6547,
            "Subdivision": "Ontario",
            "CausedBy": {
                "Status": "ACTIVE",
                "Networks": [
                    {
                        "AutonomousSystemNumber": 16509,
                        "NetworkName": "Amazon.com"
                    }
                ],
                "NetworkEventType": "AWS"
            },
            "NetworkName": "DIGITALOCEAN-ASN"
        },
        {
            "City": "Shenzhen",
            "PercentOfClientLocationImpacted": 4.07,
            "PercentOfTotalTraffic": 0.61,
            "Country": "China",
            "Longitude": 114.0683,
            "AutonomousSystemNumber": 37963,
            "Latitude": 22.5455,
            "Subdivision": "Guangdong",
            "NetworkName": "Hangzhou Alibaba Advertising Co.,Ltd."
        },
        {
            "City": "Hangzhou",
            "PercentOfClientLocationImpacted": 2.88,
            "PercentOfTotalTraffic": 0.7799999999999999,
            "Country": "China",
            "Longitude": 120.1612,
            "AutonomousSystemNumber": 37963,
            "Latitude": 30.2994,
```

```
            "Subdivision": "Zhejiang",
            "NetworkName": "Hangzhou Alibaba Advertising Co.,Ltd."
        }
    ],
    "StartedAt": "2022-06-20T01:34:20Z",
    "ImpactType": "PERFORMANCE",
    "PercentOfTotalTrafficImpacted": 1.15
}
```

モニタの一覧を表示する

CLI を使用してアカウント内のすべてのモニタを一覧表示するには、list-monitors コマンドを実行します。

```
aws internetmonitor list-monitors
```

```
{
    "Monitors": [
        {
            "MonitorName": "TestMonitor",
            "ProcessingStatus": "OK",
            "Status": "ACTIVE"
        }
    ],
    "NextToken": " zase12"
}
```

モニタを編集する

CLI を使用してモニタに関する情報を更新するには、update-monitor コマンドを使用して更新するモニタの名前を指定します。モニタリングするトラフィックの割合や、モニタリングする都市ネットワーク数の上限の更新、Internet Monitor がトラフィックをモニタリングするために使用するリソースの追加または削除、モニタリングステータスの ACTIVE から INACTIVE (またはその逆) への変更を行うことができます。モニタの名前は変更できないことに注意してください。

update-monitor の呼び出しに対する応答では、MonitorArn と Status だけを返します。

以下に、update-monitor コマンドを使用してモニタリングする都市ネットワークの最大数を 50000 に変更する方法の例を示します。

```
aws internetmonitor update-monitor --monitor-name "TestMonitor" --max-city-networks-to-
monitor 50000
```

```
{
    "MonitorArn": "arn:aws:internetmonitor:us-east-1:111122223333:monitor/TestMonitor",
    "Status": " ACTIVE "
}
```

次の例で、リソースを追加および削除する方法を示します。

```
aws internetmonitor update-monitor --monitor-name "TestMonitor" \
    --resources-to-add "arn:aws:ec2:us-east-1:111122223333:vpc/vpc-11223344556677889" \
    --resources-to-remove "arn:aws:ec2:us-east-1:111122223333:vpc/vpc-2222444455556666"
```

```
{
    "MonitorArn": "arn:aws:internetmonitor:us-east-1:111122223333:monitor/TestMonitor",
    "Status": "ACTIVE"
}
```

次の例で、update-monitor コマンドを使用してモニタのステータスを INACTIVE に変更する方法を示します。

```
aws internetmonitor update-monitor --monitor-name "TestMonitor" --status "INACTIVE"
```

```
{
    "MonitorArn": "arn:aws:internetmonitor:us-east-1:111122223333:monitor/TestMonitor",
    "Status": "INACTIVE"
}
```

モニタを削除する

CLI の delete-monitor コマンドでモニタを削除できます。まず、モニタを無効化する必要があります。そのためには、update-monitor コマンドを使用してステータスを INACTIVE に変更します。get-monitor コマンドを使用してステータスをチェックし、モニタが無効化されたことを確認します。

モニタのステータスが INACTIVE であれば、 CLI を使用して delete-monitor コマンドを実行し、そのモニタを削除できます。delete-monitor の呼び出しが成功した場合の応答は空です。

```
aws internetmonitor delete-monitor --monitor-name "TestMonitor"
```

```
{}
```

Internet Monitor ダッシュボードを使用したモニタリングと最適化

このセクションでは、Amazon CloudWatch Internet Monitor ダッシュボードで情報をフィルタリングおよび表示して、AWS アプリケーションのインターネットトラフィックと設定に関するインサイトを視覚化したり、インサイトを得たりする方法を説明します。

アプリケーションのインターネットパフォーマンスおよびアベイラビリティをモニタリングするモニターを作成すると、Amazon CloudWatch Internet Monitor は、クライアントのロケーションとネットワーク (都市ネットワーク) のペアに関するインターネット測定値を含む CloudWatch ログと、アプリケーション、各 AWS リージョン、エッジロケーションへのトラフィックに関して集計した CloudWatch メトリクスを発行します。Internet Monitor から得られるこの情報を基に、さまざまな手法でフィルタリングや調査を行ったり、そこからアクションについての提案を得たりすることが可能です。

使用を開始するには、CloudWatch コンソールの [アプリケーションのモニタリング] の下で [Internet Monitor] を選択します。

このセクションでは主に、AWS Management Console を使用して Internet Monitor メトリクスをフィルタリングおよび表示する方法について説明します。あるいは、AWS CLI または SDK で Internet Monitor API オペレーションを使用して、CloudWatch Logs ファイルに保存されている Internet Monitor イベントを直接操作することもできます。詳細については、「モニターの使用と測定に関する情報」を参照してください。API オペレーションの使用の詳細については、「Amazon CloudWatch Internet Monitor での CLI の使用例」および「Amazon CloudWatch Internet Monitor API リファレンス」を参照してください。

Internet Monitor ダッシュボードには 3 つのタブがあります。

- [Overview] (概要) タブでは、アプリケーションのパフォーマンスと可用性に関する現在と過去の情報、およびクライアントロケーションに影響するヘルスイベントを確認できます。
- 次の [Historical Explorer] タブでは、ロケーション、ASN、日付などでフィルタリングできます。また、グラフを使って、インターネットトラフィックに関するメトリクスの推移を可視化できます。
- [トラフィックインサイト] タブでは、モニタリング回数が上位のトラフィックに関する情報を、各種のカスタマイズ可能な方法で要約して表示できます。加えて、さまざまなロケーションと ASN

のペアについて、パフォーマンスを改善するための最適な設定に関する提案を取得することができます。トラフィックのルーティング方法や使用する AWS のリソースを変更すると、Internet Monitor はトラフィックパターンと過去のパフォーマンスに基づいて、アプリケーションでのパフォーマンスの改善を予測します。また、モニター用に選択したアプリケーショントラフィックの割合に基づいて、モニタリングカバレッジに含まれる都市ネットワークの数を比較するグラフを表示することもできます。

また、Internet Monitor ではトラフィックに関する測定値を含むログファイルを生成してパブリッシュするため、コンソール内の他の CloudWatch ツールを使用することができます。例えば、Internet Monitor によってパブリッシュされた CloudWatch Contributor Insights、CloudWatch メトリクス、CloudWatch Logs Insights などのデータをさらに可視化することができます。詳細については、「CloudWatch ツールと Internet Monitor のクエリインターフェイスによるデータの調査」を参照してください。

次のセクションで、Internet Monitor を使用してパフォーマンスと可用性の測定値を調査する方法について説明します。

トピック

- Amazon CloudWatch Internet Monitor ([Overview] (概要) タブ) によるパフォーマンスと可用性のリアルタイムでの追跡
- Amazon CloudWatch Internet Monitor ([Historical Explorer] タブ) でデータをフィルタリングおよび表示する
- Amazon CloudWatch Internet Monitor (トラフィックインサイトのタブ) でアプリケーションのパフォーマンスを改善するためのインサイトを取得する

Amazon CloudWatch Internet Monitor ([Overview] (概要) タブ) によるパフォーマンスと可用性のリアルタイムでの追跡

CloudWatch コンソールの [Internet Monitor] にある [概要] タブを使用して、モニタが追跡しているトラフィックのパフォーマンスと可用性の概要を取得します。このタブはまた、アプリケーションのグローバルトラフィックを可視化する際に役立つトラフィッククラスターと共に、インターネットトラフィックの概要マップ、およびヘルスイベントのロケーションと影響を表示できます。

ヘルススコア

[ヘルススコア] のグラフには、グローバルトラフィックのパフォーマンスと可用性に関する情報が表示されます。AWS は、さまざまな ASN や AWS サービスの地理的位置間のネットワークト

ラフィックに対する、インターネットのパフォーマンスと可用性に関する膨大な履歴データを保持しています。Internet Monitor は、グローバルネットワークのフットプリントから AWS により取得されたこの接続データを使用して、インターネットトラフィックのパフォーマンスと可用性に関するベースラインを計算します。これは、AWS の社内でインターネットの稼働時間と可用性を監視するために使用しているものと同じデータです。

これらの測定値をベースラインとして使用し、その値と比較してアプリケーションのパフォーマンスと可用性が低下した場合に、 Internet Monitor が検出できるようにしています。その情報はパフォーマンススコアと可用性スコアとして報告され、パフォーマンスの低下が発生したことをユーザーが認識しやすくしています。詳細については、「CloudWatch ツールと Internet Monitor のクエリインターフェイスによるデータの調査」を参照してください。

[ヘルススコア] のグラフは、ユーザーが選択した期間に発生したヘルスイベントを示します。ヘルスイベントが発生すると、グラフ上のパフォーマンスまたは可用性を示す線が下降します。イベントを選択すると、その詳細とともに、イベントが継続した時間に関する日付と時刻の情報を示す帯がグラフ上に表示されます。

また、各データポイントについてログファイルに直接アクセスすることで、これらのメトリクスを確認することもできます。[アクション] メニューで、[CloudWatch Logs を表示] を選択します。

インターネットトラフィックの概要

[インターネットトラフィックの概要] マップには、ユーザーがアプリケーションにアクセスした場所や ASN に固有のインターネットトラフィックとヘルスイベントが表示されます。マップ上で灰色に表示されている国/地域は、アプリケーションへのトラフィックが存在する国/地域です。

マップ上の各円には、選択した期間に発生した地域のヘルスイベントが示されます。Internet Monitor は、AWS でホストされているリソースの 1 つと、ユーザーがアプリケーションにアクセスしている都市ネットワークとの間の接続に関する問題を特定のしきい値で検出すると、ヘルスイベントを作成します。マップ上の円を選択すると、その場所のヘルスイベントに関する詳細が表示されます。加えて、マップの下に表示されている [Health events] (ヘルスイベント) テーブルには、ヘルスイベントが発生しているクラスターについての詳細な情報が表示されます。

Internet Monitor は、あるイベントがアプリケーションに対し、重大でグローバルな影響を与えると判断すると、それを解決するためにモニタ内にヘルスイベントを作成します。選択した期間にヘルスイベントが発生しても、クライアントロケーションにおけるトラフィックへの影響がしきい値を超えなかった場合、マップは空になることに注意してください。詳細については、「Internet Monitor がヘルスイベントを作成して解決するタイミング」を参照してください。

ヘルスイベントのしきい値を変更する

Internet Monitor がアプリケーションのヘルスイベントを作成する方法とタイミングについては、いくつかのオプションを設定できます。変更するには、[しきい値の更新] を選択します。

Internet Monitor がヘルスイベントを作成するためにトリガーする全体的なしきい値を変更できます。デフォルトのヘルスイベントのしきい値は、パフォーマンススコアとアベイラビリティスコアの両方で 95% です。つまり、アプリケーションの全体的なパフォーマンスまたは可用性のスコアが 95% 以下に低下すると、Internet Monitor はヘルスイベントを作成します。全体的なしきい値について、ヘルスイベントは、単一の大きな問題によってトリガーされることも、複数の小さな問題が組み合わさってトリガーされることもあります。

また、ローカル (都市ネットワーク) のしきい値を変更し、全体的な影響レベルの割合と組み合わせて、ヘルスイベントをトリガーすることもできます。例えば、1 つ以上の都市ネットワーク (ロケーションと ASN、一般的には ISP) のスコアがしきい値を下回ったときにヘルスイベントが発生するしきい値を設定すると、トラフィックの少ない場所で問題が発生したときなどを把握できます。

追加のローカルしきい値オプションは、可用性またはパフォーマンススコアのローカルしきい値と連携して機能します。2 番目の要素は、Internet Monitor がローカルしきい値に基づいてヘルスイベントを作成する前に、影響を受ける必要があるトラフィック全体の割合です。

全体的なトラフィックとローカルトラフィックのしきい値オプションを設定することで、アプリケーションの使用状況やニーズに合わせて、ヘルスイベントが作成される頻度を微調整できます。ローカルのしきい値を低く設定すると、アプリケーションやその他の設定したしきい値の設定値によっては、通常より多くのヘルスイベントが作成されることに注意してください。

要約すると、パフォーマンススコア、可用性スコア、またはその両方について、ヘルスイベントのしきい値を次の方法で設定できます。

- ヘルスイベントをトリガーするには、さまざまなグローバルしきい値を選択します。
- ヘルスイベントをトリガーするには、さまざまなローカルしきい値を選択します。このオプションでは、Internet Monitor がイベントを作成する前に超える必要があるアプリケーション全体への影響のパーセンテージを変更することもできます。
- ローカルしきい値に基づくヘルスイベントのトリガーをオフにするか、ローカルしきい値オプションを有効にするかを選択します。

パフォーマンススコア、可用性スコア、またはその両方のオプションを設定することもできます。オプションを組み合わせて設定することも、そのうちの 1 つだけを設定することもできます。

パフォーマンススコア、可用性スコア、あるいはその両方のしきい値やその他の設定オプションを更新するには、以下の手順を実行してください。

しきい値設定オプションを変更するには

1. AWS Management Console で CloudWatch に移動し、左側のナビゲーションペインで [Internet Monitor] を選択します。
2. [概要] タブの [ヘルスイベントのタイムライン] セクションで、[しきい値の更新] を選択します。
3. 開いたダイアログページで、しきい値に必要な新しい値とオプション、および Internet Monitor がヘルスイベントを作成するトリガーとなるその他のオプションを選択します。以下のいずれかを実行できます。
   - [可用性スコアのしきい値]、[パフォーマンススコアのしきい値]、またはその両方の新しい値を選択します。

     各設定のセクションのグラフには、アプリケーションの現在のしきい値の設定と、アベイラビリティまたはパフォーマンスに関する最近のヘルスイベントの実際のスコアが表示されます。典型的な値を表示すると、しきい値を変更する必要がある値を知るのに役立ちます。

     ヒント: より大きなグラフを表示して時間枠を変更するには、グラフの右上にあるエキスパンダーを選択します。
   - 可用性またはパフォーマンス、あるいはその両方について、ローカルのしきい値をオンにするかオフにするかを選択します。オプションを有効にすると、Internet Monitor にヘルスイベントを作成させるときのしきい値と影響レベルを設定できます。
4. しきい値オプションを設定したら、[ヘルスイベントのしきい値の更新] を選択して更新を保存します。

ヘルスイベントの仕組みの詳細については、「Internet Monitor がヘルスイベントを作成および解決する場合」を参照してください。

ヘルスイベントテーブル

[ヘルスイベント] テーブルには、ヘルスイベントの影響を受けたクライアントロケーションと、そのイベント自体に関する情報が一覧表示されます。この テーブルには、以下の列があります。

	説明
クライアントロケーション	イベントの影響を受けたエンドユーザーのうち、レイテンシーが増加したり、可用性が低下したりしたユーザーの場所。 Internet Monitor のクライアントのロケーションの精度の詳細については、「Internet Monitor の位置情報と精度」を参照してください。
トラフィックインパクト	そのイベントによって、レイテンシーの増加や可用性の低下など、どの程度の影響が生じたか。レイテンシーについては、(該当するクライアントロケーションから、このクライアントネットワークを使用する AWS のロケーションに向かう) 一般的なトラフィックのパフォーマンスと比較して、イベント中に増加した割合が表示されます。
クライアントネットワーク	トラフィックが通過したネットワークです。通常これは、ネットワークトラフィックのインターネットサービスプロバイダー (ISP) または AS 番号 (ASN) のことです。
AWS のロケーション	対象のネットワークトラフィックの AWS ロケーションを指します。AWS リージョン もしくはインターネットのエッジロケーションです。
影響タイプ	ヘルスイベントが与えた影響の種類です。ヘルスイベントは通常、レイテンシーの増加 (パフォーマンスの問題)、または到達可能性の低下 (可用性の問題) によって引き起こされます。 影響タイプをクリックすると、障害の原因を確認できる場合があります。可能な場

	説明
	合、Internet Monitor はヘルスイベントの発生源を分析して、原因が AWS にあるのか、それとも ASN (インターネットサービスプロバイダ) によるものなのかを判断します。 この分析は、イベントが解決された後も継続されることに注意してください。新しい情報があると、Internet Monitor はイベントを最大 1 時間で更新できます。

[ヘルスイベント] テーブルでいずれかのクライアントロケーションを選択すると、そのロケーションのヘルスイベントに関する詳細が表示されます。例えば、イベントの開始および終了の時刻や、ローカルトラフィックへの影響を確認できます。

ネットワークパスの可視化

障害解析が完了すると、[ネットワークパスの可視化] にフルネットワークパスが表示されます。フルパスには、クライアントとロケーションのペアについて、AWS のロケーションとクライアントの間にある各ノードが表示されます。この表示は、ヘルスイベントに関するアプリケーションのネットワークパスに沿っています。

Internet Monitor が障害の原因を特定すると、赤い破線の丸が付きます。障害は、ASN (通常はインターネットサービスプロバイダー (ISP)) が原因である場合もあれば、AWS が原因である場合もあります。障害の原因が複数ある場合は、複数のノードが丸で囲まれます。

Amazon CloudWatch Internet Monitor ([Historical Explorer] タブ) でデータをフィルタリングおよび表示する

CloudWatch コンソールの Internet Monitor にある [履歴エクスプローラー] タブを使用して、CloudWatch Logs ファイル内のアプリケーションに関するデータをフィルタリングして表示します。Internet Monitor は、アプリケーションに固有の CloudWatch Log に、可用性、パフォーマンス、モニタリング対象転送バイト数 (またはクライアント接続数、WorkSpaces ディレクトリの場合のみ)、および監視対象の AWS リージョン 内の都市ネットワークの往復時間の測定値をパブリッシュします。

> **Note**
>
> Internet Monitor は、インターネット測定値を 5 分ごとに CloudWatch Logs にパブリッシュします。対象は、各モニタにトラフィックを送信する (トラフィック量で) 上位 500 の都市ネットワーク (つまり、クライアントロケーションと ASN、通常はインターネットサービスプロバイダー (ISP)) です。必要に応じて、モニタリング対象の都市ネットワークすべてのインターネット測定値を Amazon S3 バケットにパブリッシュする選択ができます (ただし、サービスの上限数は 500,000 の都市ネットワークです)。詳細については、「Amazon CloudWatch Internet Monitor で Amazon S3 にインターネット測定値をパブリッシュする」を参照してください。

アプリケーションデータの調査を開始するには、期間を選択します。次に、特定の地理的位置 (都市など) や、ログファイルに適用するその他のフィルターを (オプションで) 選択します。Internet Monitor は、インターネット測定ログ内のデータにフィルターを適用します。このログファイルは、アプリケーショントラフィックに関して都市ネットワーク向けに Internet Monitor がパブリッシュしたものです。これにより、時間の経過に伴うアプリケーションの推移をグラフで確認できます。このグラフには、パフォーマンススコア、可用性スコア、モニタリング対象転送バイト数 (VPC、Network Load Balancers と CloudFront ディストリビューションの場合) またはクライアント接続数 (WorkSpaces ディレクトリの場合)、往復時間 (RTT) が表示されます。

グラフの下にある [All events] (すべてのイベント) のテーブルには、対象のアプリケーショントラフィックについてフィルタリングされたヘルスイベントと、各イベントに関する情報が表示されます。情報には以下の列が含まれます。

	説明
イベントの開始	ヘルスイベントが開始された時刻。
ステータス	イベントがまだアクティブであるか、解決済みであるか。
クライアントロケーション	イベントの影響を受けたエンドユーザーのうち、レイテンシーが増加したり、パフォーマンスが低下したりしたエンドユーザーの場所。

	説明
	Internet Monitor のクライアントの口ケーションの精度の詳細については、「Internet Monitor の位置情報と精度」を参照してください。
トラフィックインパクト	ヘルスイベントの対象の場所における、イベントの影響の重み。これは例えば、レイテンシーへの影響などで、クライアント口ケーションからクライアント ASN (通常はインターネットサービスプロバイダー (ISP)) を経由して、AWS の口ケーションに向かうトラフィックの標準的なパフォーマンスと比較されます。同様に、可用性に影響するイベントの場合は、可用性に対する影響の程度が、クライアント ASN を経由した AWS 口ケーションにおけるこのクライアント口ケーションの標準的な可用性と比較されます。
イベント期間	イベントが継続した長さです。Internet Monitor は、ヘルスイベントが影響を与えるアプリケーションのクライアント口ケーションが 5% (合計) 以下になった時点で、そのイベントを終了します。
クライアント ISP	ASN、通常はインターネットサービスプロバイダー (ISP) のことで、ネットワークトラフィックのキャリアでした。
サービス口ケーション	ネットワークトラフィックの発信元のサービスがある口ケーションであり、AWS リージョンもしくはエッジ口ケーションです。

また、各データポイントのログに直接アクセスして、アプリケーションの測定値を確認することもできます。[アクション] メニューで、[CloudWatch Logs を表示] を選択します。測定イベントは作成された時点でアカウントにパブリッシュされるため、これらのデータに基づいて他の CloudWatch ダッシュボードやアラームを作成することもできます。詳細については、Amazon CloudWatch Internet

Monitor (トラフィックインサイトのタブ) でアプリケーションのパフォーマンスを改善するためのインサイトを取得するおよびAmazon CloudWatch Internet Monitor でアラームを作成するを参照してください。

Internet Monitor 測定値とメトリクスを確認して分析したり、それらのデータに基づいてダッシュボードやアラームを作成したりすることに加えて、アプリケーションのパフォーマンスを改善する方法を理解するために Internet Monitor を使用することもできます。[トラフィックインサイト] タブで、このオプションを探すのに役立つ複数の方法が提供されます。詳細については、[トラフィックインサイト] タブの [トラフィック最適化のための提案事項] を参照してください。また、具体的な例については、「Internet Monitor 使用例」の章を参照してください。

Amazon CloudWatch Internet Monitor (トラフィックインサイトのタブ) でアプリケーションのパフォーマンスを改善するためのインサイトを取得する

CloudWatch コンソールの [Internet Monitor] にある [トラフィックインサイト] タブを使用して、アプリケーションのトラフィック (量で) 上位の概要情報を確認します。アプリケーショントラフィックは複数の方法でフィルタリングやソートができます。下へスクロールして、アプリケーションのさまざまな設定の組み合わせを選択します。そこでは Internet Monitor が、最初のバイトまでの時間 (TTFB) のパフォーマンスを最速にするための最良の代替手段を提案しています。

Internet Monitor は、インターネット測定値を 5 分ごとに CloudWatch Logs にパブリッシュします。対象は、各モニタにトラフィックを送信する (トラフィック量で) 上位 500 の都市ネットワーク (つまり、クライアントロケーションと ASN、通常はインターネットサービスプロバイダー (ISP)) です。必要に応じて、モニタリング対象の都市ネットワークすべてのインターネット測定値を Amazon S3 バケットにパブリッシュする選択ができます (ただし、サービスの上限数は 500,000 の都市ネットワークです)。詳細については、「Amazon CloudWatch Internet Monitor で Amazon S3 にインターネット測定値をパブリッシュする」を参照してください。

上位のトラフィックサマリー

まず、特定の期間におけるアプリケーション全体のトラフィックとパフォーマンスの概要を、クライアントのロケーションでフィルタリングして表示します。また、トラフィック量による上位 (あるいは下位) のクライアントロケーションにおけるアプリケーションのパフォーマンスを確認でき、さまざまな方法でフィルタリングやソートもできます。例えば、詳細度 (都市、行政区画、国/地域、都市圏など) や合計トラフィック量、最初のバイトまでの時間 (TTFB) における平均時間などの要素でソートできます。

Internet Monitor のクライアントのロケーションの精度の詳細については、「Internet Monitor の位置情報と精度」を参照してください。

> **Note**
>
> 使用するフィルターはページ全体に適用されるため、総トラフィック量の概要グラフや情報に含まれる都市ネットワークに影響を与えます。また次の [トラフィック最適化の提案] セクションに含まれる都市ネットワークにも影響を与えます。

トラフィック最適化のための提案事項

[トラフィック最適化の提案] セクションには、トラフィックのモニタリング対象都市ネットワーク (ロケーションと ASN、インターネットサービスプロバイダー) のフィルターセットと、各都市ネットワークのクライアントの総トラフィック量が表示されます。テーブル内に入力されたデータは、ページ上部にある [トラフィックインサイト] で選択したアプリケーショントラフィックのフィルターに基づいています。デフォルトは、トラフィック量の多い上位 10 都市です。固有の都市ネットワークペアごとに入力データが存在するため、通常、テーブルには 10 行以上が表示されます。つまり、クライアントがアプリケーションにアクセスする場所 (都市) と ASN (ネットワークプロバイダ) の組み合わせごとに 1 行ずつ表示されることになります。たとえば、ダラス、テキサス、米国、Comcast などとなります。

> **Note**
>
> 監視しているすべての都市ネットワークに対するトラフィック最適化のための提案事項を確認するには、CloudWatch Insights で直接クエリを実行します。このページの都市ネットワークのリストを制限する地理的な詳細度フィルターを含まないクエリ例については、「Amazon CloudWatch Internet Monitorでの CloudWatch Logs Insights の使用」を参照してください。

このセクションでは別のオプションを選択します。Amazon EC2、CloudFront またはその両方です。これにより、様々な AWS リージョンにあるこれらのサービスでアプリケーションを使用した場合のクライアントの最初のバイトまでの平均時間 (TTFB) の予測値を、現在の TTFB と比較することができます。TTFB 計算の詳細については、「TTFB とレイテンシーに対する AWS の計算」を参照してください。

さまざまなオプションを選択し、結果をテーブルに表示することが、クライアントのパフォーマンスを改善できるような設定とデプロイの計画を始めるのに役立ちます。データを表示できない場合は、列の値ではなくダッシュ (-) が表示されることがあることに注意してください。パ

フォーマンスを向上させる方法の具体例については、「ゲーム体験を向上させるための Amazon CloudWatch Internet Monitor の使用」を参照してください。

開始するには、例えば特定の都市ネットワーク (クライアントロケーションと ASN のペア) について、EC2 または CloudFront のオプション、あるいはその両方を選択してみてください。テーブルに記載された各都市ネットワークについて、Internet Monitor は、そのオプションを (特定の AWS リージョン を介して) 使用したトラフィックルーティングの選択に基づいて、現在の設定と比較した TTFB のパフォーマンス改善の可能性を示します (完全を期すため、このリストには既に最適化されたルートも含まれていることに注意してください)。例えば、us-west-2 経由で EC2 ルーティングを使用している現在の設定での TTFB が 100 ミリ秒であるのに対し、us-east-1 経由で EC2 ルーティングを使用した場合の予測平均 TTFB が 50 ミリ秒となることを確認できます。そのため、us-west-2 経由のルーティングを検討することもできます。

別の例として、ある 1 つのクライアントのロケーションと ASN では、EC2 を選択しても顕著なパフォーマンスの違いは得られないが、同じリージョンを使用して CloudFront を選択すると、TTFB を一定量低減できることがわかる場合もあります。このクライアントロケーションと ASN では、アプリケーションの前に CloudFront ディストリビューションを追加するとパフォーマンスが改善する可能性があり、したがって、これを試してみる価値はあると言えます。

CloudWatch ツールと Internet Monitor のクエリインターフェイスによるデータの調査

Amazon CloudWatch Internet Monitor ダッシュボードを使用してアプリケーションのパフォーマンスと可用性を可視化することに加えて、Internet Monitor が生成するデータをより詳しく調べるために使用できる方法が、いくつか用意されています。これらの方法には、CloudWatch のログファイルに保存されている Internet Monitor のデータで CloudWatch ツールを使用することや、Internet Monitor のクエリインターフェイスを使用することが含まれます。ここで使用できるツールとしては、CloudWatch Logs Insights、CloudWatch Metrics、CloudWatch Contributor Insights、Amazon Athena などがあります。必要に応じて、ダッシュボードだけでなくこれらのツールの一部またはすべてを使用し、Internet Monitor のデータを調べることができます。

Internet Monitor はアプリケーションと各 AWS リージョン へのトラフィックに関する CloudWatch メトリクスを集計し、また、総合的なトラフィックへの影響、可用性、往復時間などのデータも集計します。このデータは CloudWatch Logs に公開され、Internet Monitor のクエリインターフェイスでも使用できます。調査可能な地理的粒度や、その他の側面についての詳細は、その情報によって異なります。

Amazon CloudWatch Internet Monitor は、5 分間隔でモニターのデータを公開し、さまざまな方法でデータを利用できるようにします。Internet Monitor のデータにアクセスするシナリオと、そこで収集されるデータの特徴を次の表に示します。

機能	CloudWatch ログ	S3 へのエクスポート	クエリインターフェイス	CloudWatch ダッシュボード
デフォルトで有効	はい	いいえ	はい	はい
データ収集の対象となる都市ネットワークの数	上位 500 件 (下記の注記を参照)	すべて	すべて	すべて
データ保持	ユーザー制御	ユーザー制御	30 日間	30 日間
データ収集の地理的粒度	すべて (都市ネットワーク、都市部とネットワーク、行政区画とネットワーク、国とネットワーク)	都市ネットワーク	すべて (都市ネットワーク、都市部とネットワーク、行政区画とネットワーク、国とネットワーク)	すべて (都市ネットワーク、都市部とネットワーク、行政区画とネットワーク、国とネットワーク)
データのクエリとフィルタリングを行う方法	Amazon CloudWatch Internet Monitor での CloudWatch Logs Insights の使用	Amazon Athena を使用して、Amazon S3 ログファイルのインターネット測定値をクエリする	Amazon CloudWatch Internet Monitor クエリインターフェイスの使用	Internet Monitor ダッシュボードを使用したモニタリングと最適化

注:上位 500 件の測定値は都市ネットワークで取得されます。上位 250 件は都市部とネットワーク、上位 100 件は行政区画とネットワーク、上位 50 件は国とネットワークで取得されます。

この章では、CloudWatch ツールまたは Internet Monitor クエリインターフェイスを使用してデータをクエリし調査する方法を、それぞれの方法の例とともに説明します。

内容

Amazon CloudWatch Internet Monitorでの CloudWatch Logs Insights の使用

Amazon CloudWatch Internet Monitor は、可用性と往復時間についての詳細な測定値を、CloudWatch Logs に対しパブリッシュします。CloudWatch Logs Insights クエリを使用して、特定の都市または地域 (クライアントロケーション)、クライアント ASN (ISP)、および AWS のソースロケーションに関するログのサブセットをフィルタリングできます。

Internet Monitor のクライアントのロケーションの精度の詳細については、「Internet Monitor の位置情報と精度」を参照してください。

このセクションの例は、CloudWatch Logs Insights クエリを作成して、独自のアプリケーショントラフィックの測定値やメトリクスについて詳細を知るのに役立ちます。CloudWatch Logs Insights でこれらの例を使用する場合は、*monitorName* を独自のモニタ名に置き換えます。

トラフィック最適化の提案を表示する

Internet Monitor の [トラフィックインサイト] タブでは、ロケーション別にフィルタリングされたトラフィック最適化の提案を表示できます。そのタブ内に存在する [トラフィック最適化の提案] セクションに表示されている情報と同じ情報を、ロケーションの詳細度フィルターなしで表示するには、次の CloudWatch Logs Insights クエリを使用できます。

1. AWS Management Console の中で、CloudWatch Logs Insights に移動します。
2. [Log Group] (ロググループ) で、`/aws/internet-monitor/`*monitorName*`/byCity` および `/aws/internet-monitor/`*monitorName*`/byCountry` を選択した後に、時間範囲を指定します。
3. 以下のクエリを追加した後に、それらを実行します。

```
fields @timestamp,
clientLocation.city as @city, clientLocation.subdivision as @subdivision,
 clientLocation.country as @country,
```

```
`trafficInsights.timeToFirstByte.currentExperience.serviceName` as @serviceNameField,
concat(@serviceNameField, ` (`, `serviceLocation`, `)`) as @currentExperienceField,
concat(`trafficInsights.timeToFirstByte.ec2.serviceName`, ` (`,
 `trafficInsights.timeToFirstByte.ec2.serviceLocation`, `)`) as @ec2Field,
`trafficInsights.timeToFirstByte.cloudfront.serviceName` as @cloudfrontField,
concat(`clientLocation.networkName`, ` (AS`, `clientLocation.asn`, `)`) as @networkName
| filter ispresent(`trafficInsights.timeToFirstByte.currentExperience.value`)
| stats avg(`trafficInsights.timeToFirstByte.currentExperience.value`) as @averageTTFB,
avg(`trafficInsights.timeToFirstByte.ec2.value`) as @ec2TTFB,
avg(`trafficInsights.timeToFirstByte.cloudfront.value`) as @cloudfrontTTFB,
sum(`bytesIn` + `bytesOut`) as @totalBytes,
latest(@ec2Field) as @ec2,
latest(@currentExperienceField) as @currentExperience,
latest(@cloudfrontField) as @cloudfront,
count(*) by @networkName, @city, @subdivision, @country
| display @city, @subdivision, @country, @networkName, @totalBytes, @currentExperience,
 @averageTTFB, @ec2, @ec2TTFB, @cloudfront, @cloudfrontTTFB
| sort @totalBytes desc
```

インターネットの可用性と RTT (p50、p90、p95) を表示する

トラフィックでのインターネットの可用性と往復時間 (p50、p90、p95) を表示するには、以下の CloudWatch Logs Insights クエリを使用できます。

エンドユーザーの地域: 米国イリノイ州シカゴ

エンドユーザーネットワーク (ASN): AS7018

AWS のサービスの拠点: 米国東部 (バージニア北部) リージョン

ログを表示するには、次の操作を行います。

1. AWS Management Console の中で、CloudWatch Logs Insights に移動します。
2. [Log Group] (ロググループ) で、`/aws/internet-monitor/`*`monitorName`*`/byCity` および `/aws/internet-monitor/`*`monitorName`*`/byCountry` を選択した後に、時間範囲を指定します。
3. 以下のクエリを追加した後に、それらを実行します。

このクエリは、選択した期間において、イリノイ州シカゴの AS7018 から米国東部 (バージニア北部) リージョンに接続していたユーザーの、すべてのパフォーマンスデータを返します。

```
fields @timestamp,
```

```
internetHealth.availability.experienceScore as availabilityExperienceScore,
internetHealth.availability.percentageOfTotalTrafficImpacted as
 percentageOfTotalTrafficImpacted,
internetHealth.performance.experienceScore as performanceExperienceScore,
internetHealth.performance.roundTripTime.p50 as roundTripTimep50,
internetHealth.performance.roundTripTime.p90 as roundTripTimep90,
internetHealth.performance.roundTripTime.p95 as roundTripTimep95
 | filter clientLocation.country == `United States`
 and clientLocation.city == `Chicago`
 and serviceLocation == `us-east-1`
 and clientLocation.asn == 7018
```

詳細については、「CloudWatch Logs Insights を使用したログデータの分析」を参照してください。

Amazon CloudWatch Internet Monitor での Contributor Insights の使用

CloudWatch Contributor Insights は、アプリケーションで上位のクライアントロケーションとネットワーク (ASN またはインターネットサービスプロバイダー) を特定するのに役立ちます。Amazon CloudWatch Internet Monitor で役立つルールの使用開始には、以下の Contributor Insights ルール例を使用できます。詳細については、「Contributor Insights ルールの作成」を参照してください。

Internet Monitor のクライアントのロケーションの精度の詳細については、「Internet Monitor の位置情報と精度」を参照してください。

> **Note**
>
> Internet Monitor は 5 分ごとにデータを公開するので、Contributor Insights ルールを設定した後は、グラフが表示される間隔を 5 分に調整する必要があります。

可用性の影響を受ける上位のロケーションと ASN を表示する

可用性の低下によって影響を受けた上位のクライアントロケーションと ASN を表示するには、構文エディタで次の Contributor Insights ルールを使用します。*monitor-name* は、実際のモニターの名前に置き換えます。

```
{
    "Schema": {
        "Name": "CloudWatchLogRule",
        "Version": 1
    },
    "AggregateOn": "Sum",
```

```
    "Contribution": {
        "Filters": [
            {
                "Match": "$.clientLocation.city",
                "IsPresent": true
            }
        ],
        "Keys": [
            "$.clientLocation.city",
            "$.clientLocation.networkName"
        ],
        "ValueOf": "$.awsInternetHealth.availability.percentageOfTotalTrafficImpacted"
    },
    "LogFormat": "JSON",
    "LogGroupNames": [
        "/aws/internet-monitor/monitor-name/byCity"
    ]
}
```

レイテンシーの影響を受ける上位のクライアントロケーションと ASN を表示する

往復時間 (レイテンシー) の増加によって影響を受けた上位のクライアントロケーションと ASN を表示するには、構文エディタで次の Contributor Insights ルールを使用します。*monitor-name* は、実際のモニターの名前に置き換えます。

```
{
    "Schema": {
        "Name": "CloudWatchLogRule",
        "Version": 1
    },
    "AggregateOn": "Sum",
    "Contribution": {
        "Filters": [            {
                "Match": "$.clientLocation.city",
                "IsPresent": true
            }
        ],
        "Keys": [
            "$.clientLocation.city",
            "$.clientLocation.networkName"
        ],
        "ValueOf": "$.awsInternetHealth.performance.percentageOfTotalTrafficImpacted"
    },
```

```
    "LogFormat": "JSON",
    "LogGroupNames": [
        "/aws/internet-monitor/monitor-name/byCity"
    ]
}
```

トラフィックの合計パーセンテージによって影響を受けた上位のクライアントロケーションと ASN を表示する

トラフィックの合計パーセンテージによって影響を受けた上位のクライアントロケーションと ASN を表示するには、構文エディタで次の Contributor Insights ルールを使用します。*monitor-name* は、実際のモニターの名前に置き換えます。

```
{
    "Schema": {
        "Name": "CloudWatchLogRule",
        "Version": 1
    },
    "AggregateOn": "Sum",
    "Contribution": {
        "Filters": [
            {
                "Match": "$.clientLocation.city",
                "IsPresent": true
            }
        ],
        "Keys": [
            "$.clientLocation.city",
            "$.clientLocation.networkName"
        ],
        "ValueOf": "$.percentageOfTotalTraffic"
    },
    "LogFormat": "JSON",
    "LogGroupNames": [
        "/aws/internet-monitor/monitor-name/byCity"
    ]
}
```

Amazon CloudWatch Internet Monitor での CloudWatch Metrics の使用

Amazon CloudWatch Internet Monitor は、パフォーマンス、アベイラビリティ、往復時間、スループット (1 秒あたりのバイト数) に関するメトリクスをアカウントに発行します。これらのメトリ

クスは、CloudWatch コンソールの CloudWatch メトリクスで表示できます。モニタですべてのメトリクスを検索するには、CloudWatch メトリクスダッシュボードでカスタム名前空間 AWS/InternetMonitor を使用します。

メトリクスは、モニタ内の VPC、Network Load Balancers、CloudFront ディストリビューション、WorkSpaces ディレクトリへのインターネットトラフィックすべて、およびモニタリング対象の各 AWS リージョン とインターネットエッジロケーションのトラフィックすべてにわたって集計されます。各リージョンはサービスロケーションによって定義され、すべてのロケーションを指定することとも、特定のリージョン (us-east-1 など) にすることもできます。

注: 都市ネットワークとは、クライアントのロケーションと ASN (通常はインターネットサービスプロバイダー (ISP)) のことです。

Internet Monitor は以下のメトリクスを提供しています。

メトリクス	説明
PerformanceScore	パフォーマンススコアは、パフォーマンスの低下とは認識されないトラフィックの割合の推定値を表します。
AvailabilityScore	可用性スコアは、トラフィックにおいて可用性の低下だと判定されない割合の推定値を表します。
BytesIn	アプリケーションの都市ネットワークすべてにおけるアプリケーションインターネットトラフィックの受信バイト数を表します。
BytesOut	アプリケーションの都市ネットワークすべてにおけるアプリケーションインターネットトラフィックの送信バイト数を表します。
BytesInMonitored	モニタリング対象の都市ネットワークにおけるアプリケーションインターネットトラフィックの受信バイト数を表します。

メトリクス	説明
BytesOutMonitored	モニタリング対象の都市ネットワークにおけるアプリケーションインターネットトラフィックの送信バイト数を表します。
Round-trip time (RTT)	AWS リージョン、ASN (通常はインターネットサービスプロバイダーまたは ISP) や、VPC、Network Load Balancers、CloudFront ディストリビューション、WorkSpaces ディレクトリに固有の (都市などの) ロケーションの間における往復時間を表します。
CityNetworksMonitored	アプリケーションのインターネットトラフィックについて、Internet Monitor のモニタリング対象都市ネットワークの数を表します。これは、そのモニタに関する都市ネットワークの最大数として設定した上限を超えることはありません。
TrafficMonitoredPercent	モニタのアプリケーションにおけるインターネットトラフィックの合計のうち、Internet Monitor がモニタリングしている都市ネットワークに代表される (含まれる) パーセンテージを表します。そのモニタに設定した都市ネットワークの最大数を超える都市ネットワーク数でクライアントがアプリケーションにアクセスする場合、この数は 100 未満 (つまり 100% 未満) になります。
CityNetworksFor100PercentTraffic	Internet Monitor でアプリケーションのインターネットトラフィックを 100% モニタリングしたい場合に、設定すべき都市ネットワークの最大数を表します。

メトリクス	説明
CityNetworksFor99PercentTraffic	Internet Monitor でアプリケーションのインターネットトラフィックを 99% モニタリングしたい場合に、設定すべき都市ネットワークの最大数を表します。
CityNetworksFor95PercentTraffic	Internet Monitor でアプリケーションのインターネットトラフィックを 95% モニタリングしたい場合に、設定すべき都市ネットワークの最大数を表します。
CityNetworksFor90PercentTraffic	Internet Monitor でアプリケーションのインターネットトラフィックを 90% モニタリングしたい場合に、設定すべき都市ネットワークの最大数を表します。
CityNetworksFor75PercentTraffic	Internet Monitor でアプリケーションのインターネットトラフィックを 75% モニタリングしたい場合に、設定すべき都市ネットワークの最大数を表します。
CityNetworksFor50PercentTraffic	Internet Monitor でアプリケーションのインターネットトラフィックを 50% モニタリングしたい場合に、設定すべき都市ネットワークの最大数を表します。
CityNetworksFor25PercentTraffic	Internet Monitor でアプリケーションのインターネットトラフィックを 25% モニタリングしたい場合に、設定すべき都市ネットワークの最大数を表します。

Note

これらのメトリクスは、モニタの都市ネットワークの最大値として選択する値を決定するのに役立ちます。そのいくつかを使用する例については、「都市ネットワークの最大値を選択する」を参照してください。

詳細については、「Amazon CloudWatch メトリクスを使用する」を参照してください。

Amazon Athena を使用して、Amazon S3 ログファイルのインターネット測定値をクエリする

Amazon Athena を使用すると、Amazon CloudWatch Internet Monitor が Amazon S3 バケットに発行するインターネット測定値をクエリして表示できます。Internet Monitor では、モニタリング対象の都市ネットワーク (クライアントのロケーションと ASN、通常はインターネットサービスプロバイダー (ISP)) のインターネット向けトラフィック用 S3 バケットにアプリケーションのインターネット測定値を発行するオプションが用意されています。S3 に測定値をパブリッシュすることを選択するかどうかにかかわらず、Internet Monitor は、各モニタの (トラフィック量で) 上位 500 の都市ネットワークのインターネット測定値を、5 分ごとに CloudWatch Logs に自動的にパブリッシュします。

この章では、S3 ログファイルにあるインターネット測定値のテーブルを Athena に作成する手順を説明します。また、測定値をさまざまなビューで表示するクエリの例を紹介します。例えば、レイテンシーによる影響度の上位 10 の都市ネットワークをクエリできます。

Amazon Athena を使用して、Internet Monitor でインターネット測定値用のテーブルを作成する

Internet Monitor S3 ログファイルで Athena の使用を開始するには、まずインターネット測定値用のテーブルを作成します。

以下の手順に従い、S3 ログファイルに基づいて Athena にテーブルを作成します。次に、これらのインターネット測定クエリの例など、テーブルで Athena クエリを実行して、測定値に関する情報を取得できます。

Athena テーブルを作成する

1. https://console.aws.amazon.com/athena/ から Athena コンソールを開きます。
2. Athena クエリエディタでクエリステートメントを入力して、Internet Monitor のインターネット測定値を含むテーブルを生成します。LOCATION パラメータの値を、Internet Monitor のインターネット測定値が保存されている S3 バケットの場所に置き換えます。

```
CREATE EXTERNAL TABLE internet_measurements (
    version INT,
    timestamp INT,
    clientlocation STRING,
    servicelocation STRING,
    percentageoftotaltraffic DOUBLE,
    bytesin INT,
```

```
    bytesout INT,
    clientconnectioncount INT,
    internethealth STRING,
    trafficinsights STRING
)
PARTITIONED BY (year STRING, month STRING, day STRING)
ROW FORMAT SERDE 'org.openx.data.jsonserde.JsonSerDe'
LOCATION
's3://bucket_name/bucket_prefix/AWSLogs/account_id/internetmonitor/AWS_Region/'
TBLPROPERTIES ('skip.header.line.count' = '1');
```

3. ステートメントを入力して、データを読み取るためのパーティションを作成します。例えば、以下のサンプルクエリは、1 つの指定日と 1 つのロケーション用に単一のパーティションを作成します。

```
ALTER TABLE internet_measurements
ADD PARTITION (year = 'YYYY', month = 'MM', day = 'dd')
LOCATION
's3://bucket_name/bucket_prefix/AWSLogs/account_id/internetmonitor/AWS_Region/YYYY/
MM/DD';
```

4. [Run] (実行) を選択します。

インターネット測定値用の Athena ステートメントの例

以下は、テーブルを生成するステートメントの例です。

```
CREATE EXTERNAL TABLE internet_measurements (
    version INT,
    timestamp INT,
    clientlocation STRING,
    servicelocation STRING,
    percentageoftotaltraffic DOUBLE,
    bytesin INT,
    bytesout INT,
    clientconnectioncount INT,
    internethealth STRING,
    trafficinsights STRING
)
PARTITIONED BY (year STRING, month STRING, day STRING)
ROW FORMAT SERDE 'org.openx.data.jsonserde.JsonSerDe'
LOCATION 's3://internet-measurements/TestMonitor/AWSLogs/1111222233332/internetmonitor/
us-east-2/'
```

```
TBLPROPERTIES ('skip.header.line.count' = '1');
```

以下は、データを読み取るためのパーティションを作成するステートメントの例です。

```
ALTER TABLE internet_measurements
ADD PARTITION (year = '2023', month = '04', day = '07')
LOCATION 's3://internet-measurements/TestMonitor/AWSLogs/1111222233332/internetmonitor/
us-east-2/2023/04/07/'
```

Internet Monitor のインターネット測定値に使用する Amazon Athena クエリのサンプル

このセクションには、Amazon Athena で使用できるクエリの例があります。これを使用すると、Amazon S3 にパブリッシュされたアプリケーションのインターネット測定値に関する情報を取得できます。

影響を受けた上位 10 のクライアントロケーションと ASN (トラフィックの合計パーセンテージ順) をクエリする

この Athena クエリを実行すると、影響を受けた上位 10 の都市ネットワーク (トラフィックの合計パーセンテージ順) が返されます。都市ネットワークとは、クライアントロケーションと ASN (通常はインターネットサービスプロバイダー) のことです。

```
SELECT json_extract_scalar(clientLocation, '$.city') as city,
    json_extract_scalar(clientLocation, '$.networkname') as networkName,
    sum(percentageoftotaltraffic) as percentageoftotaltraffic
FROM internet_measurements
GROUP BY json_extract_scalar(clientLocation, '$.city'),
    json_extract_scalar(clientLocation, '$.networkname')
ORDER BY percentageoftotaltraffic desc
limit 10
```

影響を受けた上位 10 のクライアントロケーションと ASN (可用性順) をクエリする

この Athena クエリを実行すると、影響を受けた上位 10 の都市ネットワーク (可用性順) が返されます。都市ネットワークとは、クライアントロケーションと ASN (通常はインターネットサービスプロバイダー) のことです。

```
SELECT json_extract_scalar(clientLocation, '$.city') as city,
    json_extract_scalar(clientLocation, '$.networkname') as networkName,
    sum(
```

```
        cast(
            json_extract_scalar(
                internetHealth,
                '$.availability.percentageoftotaltrafficimpacted'
            )
        as double )
    ) as percentageOfTotalTrafficImpacted
FROM internet_measurements
GROUP BY json_extract_scalar(clientLocation, '$.city'),
    json_extract_scalar(clientLocation, '$.networkname')
ORDER BY percentageOfTotalTrafficImpacted desc
limit 10
```

影響を受けた上位 10 のクライアントロケーションと ASN (レイテンシー順) をクエリする

この Athena クエリを実行すると、影響を受けた上位 10 の都市ネットワーク (レイテンシーの影響度順) が返されます。都市ネットワークとは、クライアントロケーションと ASN (通常はインターネットサービスプロバイダー) のことです。

```
SELECT json_extract_scalar(clientLocation, '$.city') as city,
    json_extract_scalar(clientLocation, '$.networkname') as networkName,
    sum(
        cast(
            json_extract_scalar(
                internetHealth,
                '$.performance.percentageoftotaltrafficimpacted'
            )
        as double )
    ) as percentageOfTotalTrafficImpacted
FROM internet_measurements
GROUP BY json_extract_scalar(clientLocation, '$.city'),
    json_extract_scalar(clientLocation, '$.networkname')
ORDER BY percentageOfTotalTrafficImpacted desc
limit 10
```

クライアントロケーションと ASN のトラフィックのハイライトをクエリする

この Athena クエリを実行すると、都市ネットワークの可用性スコア、パフォーマンススコア、最初のバイトまでの時間などを含むトラフィックのハイライトが返されます。都市ネットワークとは、クライアントロケーションと ASN (通常はインターネットサービスプロバイダー) のことです。

```
SELECT json_extract_scalar(clientLocation, '$.city') as city,
    json_extract_scalar(clientLocation, '$.subdivision') as subdivision,
    json_extract_scalar(clientLocation, '$.country') as country,
    avg(cast(json_extract_scalar(internetHealth, '$.availability.experiencescore') as
 double)) as availabilityScore,
    avg(cast(json_extract_scalar(internetHealth, '$.performance.experiencescore') as
 double)) performanceScore,
    avg(cast(json_extract_scalar(trafficinsights,
 '$.timetofirstbyte.currentexperience.value') as double)) as averageTTFB,
    sum(bytesIn) as bytesIn,
    sum(bytesOut) as bytesOut,
    sum(bytesIn + bytesOut) as totalBytes
FROM internet_measurements
where json_extract_scalar(clientLocation, '$.city') != 'N/A'
GROUP BY
json_extract_scalar(clientLocation, '$.city'),
    json_extract_scalar(clientLocation, '$.subdivision'),
    json_extract_scalar(clientLocation, '$.country')
ORDER BY totalBytes desc
limit 100
```

Athena の詳しい使用方法については、「Amazon Athena ユーザーガイド」を参照してください。

Amazon CloudWatch Internet Monitor クエリインターフェイスの使用

AWS アプリケーションのインターネットトラフィックをより深く理解するには、Amazon CloudWatch Internet Monitor のクエリインターフェイスを使用する方法があります。クエリインターフェイスを使用するには、選択したデータフィルタを使用してクエリを作成してから、そのクエリを実行して Internet Monitor データのサブセットを返します。クエリにより返されるデータを調べると、アプリケーションがインターネット上でどのように動作しているのか分かります。

可用性とパフォーマンスのスコア、転送バイト数、往復時間、先頭バイトまでの時間 (TTFB) など、Internet Monitor がモニターでキャプチャする、すべてのメトリクスをクエリして調べることができます。

Internet Monitor はクエリインターフェイスを使用してデータを提供し、Internet Monitor コンソールのダッシュボードでそのデータを調べることができます。ダッシュボードの [Historical Explorer] タブまたは [トラフィックインサイト] タブの検索オプションを使用すると、アプリケーションのインターネットデータのクエリとフィルタリングができます。

ダッシュボードよりも柔軟にデータを調べたりフィルタリングしたりしたい場合は、AWS Command Line Interface または AWS SDK と組み合わせて Internet Monitor API オペレーションを使

うと、自分でクエリインターフェイスを使えます。このセクションでは、アプリケーションのインターネットトラフィックに関する洞察を得るために、クエリインターフェイスで使用可能なクエリの種類と、データのサブセット作成時に指定するフィルターについて説明します。

トピック

クエリインターフェースを使用する方法

クエリインターフェイスで、クエリタイプを選択し、フィルタの値を指定してクエリを作成すると、ログファイルから目的のデータのサブセットが返ってきます。その後、データサブセットをさらに絞り込んだり、並べ替えたり、レポートを作成したりできます。

このクエリプロセスの動作は、次のようになります。

1. クエリを実行すると、Internet Monitor はクエリ固有の `query ID` を返します。このセクションでは、使用可能なクエリタイプと、クエリ内のデータをフィルタリングするオプションについて説明します。この仕組みを理解するには、「クエリの例」のセクションもご覧ください。
2. GetQueryResults API オペレーションでは、クエリ ID とモニター名を指定して、クエリの結果のデータを返します。クエリの種類ごとに、異なるデータフィールドのセットが返されます。詳細については、「クエリ結果を取得する」をご参照ください。

クエリインターフェイスには、次の 3 つのクエリタイプがあります。次に示すように、ログファイルからトラフィックについての情報を、クエリタイプごとに返します。

- 測定値: 可用性スコア、パフォーマンススコア、総トラフィック、往復時間を 5 分間隔で表示します。
- 上位のロケーション: モニタリング中の上位のロケーションと ASN の組み合わせについて、可用性スコア、パフォーマンススコア、総トラフィック、最初のバイトまでの時間 (TTFB) 情報を、トラフィック量別に表示します。
- 上位のロケーションの詳細: Amazon CloudFront の TTFB、現在の設定、最もパフォーマンスの高い Amazon EC2 設定を 1 時間間隔で表示します。

これらのクエリタイプでは、次のうち 1 つまたは複数の条件を指定することで、データをさらに絞り込めます。

- AWS ロケーション:AWS ロケーションには、CloudFront または AWS リージョン、us-east-2、us-west-2などを指定できます。
- ASN: ASN を指定します。通常はインターネットサービスプロバイダー (ISP) を指定します。
- クライアントロケーション:ロケーションとして都市、地下鉄、行政区画、または国を指定します。
- 地域: 一部のクエリでは、geo を指定します。これは Top locations のクエリタイプを使用するクエリでは必須ですが、他のクエリタイプでは使用できません。フィルタパラメーターの geo を指定するタイミングについては、「クエリの例」セクションを参照してください。

データのフィルタリングに使用できる演算子は、EQUALS と NOT_EQUALS です。パラメータのフィルタリングについて詳しくは、FilterParameter API のオペレーションを参照してください。

クエリインターフェイスのオペレーションの詳細については、「Amazon CloudWatch Internet Monitor API Reference Guide」の、次の API オペレーションを参照してください。

- クエリを作成して実行するには、StartQuery API オペレーションを参照してください。
- クエリを停止するには、StopQuery API オペレーションを参照してください。
- 作成したクエリを使用してデータを返すには、GetQueryResults API オペレーションを参照してください。
- クエリのステータスを取得するには、GetQueryStatus API オペレーションを参照してください。

クエリの例

モニターのログファイルから、フィルタリングされたデータセットを取得するクエリを作成するには、StartQuery API オペレーションを使用します。クエリについて、クエリタイプとフィルタパラメータを指定します。次に、Internet Monitor クエリインターフェイス API オペレーションを使用してクエリ結果を取得すると、処理するデータのサブセットが取得されます。

クエリタイプとフィルタパラメータがどのように動作するのか、いくつかの例を挙げて説明します。

例 1

ある都市を除き、特定の国のモニターから、ログファイルのデータをすべて取得したいとしましょう。次の例は、このシナリオの StartQuery オペレーションで作成できる、クエリのフィルターパラメーターを示しています。

```
{
   MonitorName: "TestMonitor"
   StartTime: "2023-07-12T20:00:00Z"
   EndTime: "2023-07-12T21:00:00Z"
   QueryType: "MEASUREMENTS"
   FilterParameters: [
      {
       Field: "country",
       Operator: "EQUALS",
       Values: ["Germany"]
      },
      {
       Field: "city",
       Operator: "NOT_EQUALS",
       Values: ["Berlin"]
      },
    ]
}
```

例 2

別の例として、大都市圏ごとに上位の口ケーションを確認したいとしましょう。このシナリオでは、次の例で示すクエリを使用できます。

```
{
   MonitorName: "TestMonitor"
   StartTime: "2023-07-12T20:00:00Z"
   EndTime: "2023-07-12T21:00:00Z"
   QueryType: "TOP_LOCATIONS"
   FilterParameters: [
      {
       Field: "geo",
       Operator: "EQUALS",
       Values: ["metro"]
      },
    ]
}
```

例 3

例えば、口サンゼルスの大都市圏で最もよく利用されている、都市とネットワークの組み合わせを確認するとします。そのためには、geo=city を指定してから、metro を口サンゼルスに設定しま

す。これで、全体の大都市圏とネットワークではなく、ロサンゼルス大都市圏における上位の都市ネットワークを、クエリが返すようになりました。

次のクエリの例が使用できます。

```
{
   MonitorName: "TestMonitor"
   StartTime: "2023-07-12T20:00:00Z"
   EndTime: "2023-07-12T21:00:00Z"
   QueryType: "TOP_LOCATIONS"
   FilterParameters: [
      {
       Field: "geo",
       Operator: "EQUALS",
       Values: ["city"]
      },
      {
       Field: "metro",
       Operator: "EQUALS",
       Values: ["Los Angeles"]
      }
    ]
}
```

例 4

最後に、特定の行政区画 (米国の州など) の TTFB データを取得するとします。

このシナリオのクエリは、次の例のようになります。

```
{
   MonitorName: "TestMonitor"
   StartTime: "2023-07-12T20:00:00Z"
   EndTime: "2023-07-12T21:00:00Z"
   QueryType: "TOP_LOCATION_DETAILS"
   FilterParameters: [
      {
       Field: "subdivision",
       Operator: "EQUALS",
       Values: ["California"]
      },
    ]
}
```

クエリ結果を取得する

クエリを実行した後、別の Internet Monitor API オペレーション GetQueryResults を実行すると、結果のセットをクエリとともに返すことができます。GetQueryResults を実行するときに、クエリで定義したクエリ ID と、モニターの名前を指定します。GetQueryResults は指定したクエリの結果セットからデータを取得します。

クエリを実行する場合は、GetQueryResults を実行して結果を見る前に、クエリの実行を完了していたことを確認してください。GetQueryStatus API オペレーションを使用すると、クエリが完了したかどうか確認できます。クエリの Status が SUCCEEDED になったら、結果を確認してください。

クエリが完了した後は、次の情報が結果の確認に役立ちます。次のリストで説明するように、クエリの作成時に使用したクエリタイプには、それぞれログファイルに固有な、一連のデータフィールドが含まれています。

測定

measurements のクエリタイプでは、次のデータが返ります。

timestamp, availability, performance, bytes_in, bytes_out, rtt_p50, rtt_p90, rtt_p95

上位の口ケーション

top locations クエリタイプは、場所ごとにデータをグループ化し、その期間の平均データを提供します。返されるデータには、次が含まれます。

aws_location, city, metro, subdivision, country, asn, availability, performance, bytes_in, bytes_out, current_fbl, best_ec2, best_ec2_region, best_cf_fbl

city、metro および subdivision は、geo フィールドで口ケーションタイプを選択した場合にのみ返されることに注意してください。geo で指定した口ケーションタイプに応じて、次の口ケーションフィールドが返されます。

```
city = city, metro, subdivision, country
metro = metro, subdivision, country
subdivision = subdivision, country
country = country
```

上位の口ケーションの詳細

`top locations details` クエリタイプは、時間ごとにグループ化されたデータを返します。クエリは次を返します。

```
timestamp, current_service, current_fbl, best_ec2_fbl, best_ec2_region,
best_cf_fbl
```

`GetQueryResults` API オペレーションを実行すると、Internet Monitor はレスポンスで次を返します。

- クエリが返す結果を含んでいる、文字列配列のデータ。情報は `Fields` フィールドに合わせた配列で返されます。また、API 呼び出しによっても返されます。`Fields` フィールドを使用すると、`Data` リポジトリから情報を解析でき、目的に合わせて、さらに絞り込んだりソートしたりできます。
- (`Data` フィールドのレスポンスにおいて) クエリがデータを返したフィールドを一覧表示する、フィールドの配列。この配列のそれぞれの項目は、`availability_score - float` のように、名前とデータ型がペアになっています。

トラブルシューティング

クエリインターフェイス API オペレーションを使用したときに、エラーが返される場合は、Amazon CloudWatch Internet Monitor を使用するために必要なアクセス許可があることを確認してください。特に、次のアクセス許可があることを確認してください。

```
internetmonitor:StartQuery
internetmonitor:GetQueryStatus
internetmonitor:GetQueryResults
internetmonitor:StopQuery
```

これらのアクセス許可は、コンソールで Internet Monitor ダッシュボードを使用する場合に推奨される、AWS Identity and Access Management ポリシーに含まれています。詳細については、「Amazon CloudWatch Internet Monitor 用の IAM 許可」を参照してください。

Amazon CloudWatch Internet Monitor でアラームを作成する

Amazon CloudWatch アラームは、他の CloudWatch メトリクスの場合と同様に、Amazon CloudWatch Internet Monitor のメトリクスに基づいて作成できます。

例えば、Internet Monitor のメトリクス `PerformanceScore` に基づいてアラームを作成し、メトリクスが選択した値を下回ったときに通知を送信するように設定できます。Internet Monitor メトリクスのアラームは、他の CloudWatch メトリクスと同じガイドラインに従って設定します。

アラームの作成対象として選択できる Internet Monitor メトリクスの例は次のとおりです。

- PerformanceScore
- AvailabilityScore
- RoundtripTime

Internet Monitor で使用できるすべてのメトリックを確認するには、「Amazon CloudWatch Internet Monitor での CloudWatch Metrics の使用」を参照してください。

以下の手順は、CloudWatch ダッシュボードのメトリクスに移動し、PerformanceScore に対するアラームを設定する例を示しています。次に、CloudWatch の通常の手順に従って、選択したしきい値に基づいてアラームを作成し、通知を設定するか、その他のオプションを選択します。

CloudWatch メトリックスで PerformanceScore のアラームを作成するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. [メトリクス] を選択し、次に [すべてのメトリクス] を選択します。
3. `AWS/InternetMonitor` を選択して Internet Monitor をフィルタリングします。
4. [MeasurementSource, MonitorName] を選択します。
5. リストから [PerformanceScore] を選択します。
6. [アクション] にある [GraphedMetrics] タブで、ベルのアイコンを選択し、静的なしきい値に基づいてアラームを作成します。

次に、CloudWatch の通常の手順に従ってアラームのオプションを選択します。例えば、PerformanceScore が特定のしきい値を下回った場合に Amazon SNS メッセージで通知を受けるように選択することができます。別の方法として、またはそれに加えて、ダッシュボードにアラームを追加することもできます。

以下に留意してください。

- Internet Monitor メトリクスは、通常 20 分以内に計算され、公開されます。

- Internet Monitor のメトリクスに基づいてアラームを作成する場合は、アラームのルックバック期間を設定するときに、公開前にわずかな遅延があることを考慮してください。評価期間には、ルックバック期間を最低 25 分設定することをお勧めします。

Internet Monitor で CloudWatch アラームを使用する詳細については、ブログ記事の「Using Amazon CloudWatch Internet Monitor for enhanced internet observability」を参照してください。

CloudWatch アラームを作成する時のオプションの詳細については、「静的しきい値に基づいて CloudWatch アラームを作成する」を参照してください。

Amazon EventBridge での Amazon CloudWatch Internet Monitor の使用

ネットワークの問題について Amazon CloudWatch Internet Monitor が作成するヘルスイベントは、Amazon EventBridge を使用してパブリッシュされます。これにより、アプリケーションでエンドユーザーエクスペリエンスが損なわれた際に、通知を送信することができます。

EventBridge を使用して Internet Monitor のヘルスイベントを処理するには、ここに示すガイダンスに従います。

EventBridge で Internet Monitor のルールを設定するには

1. AWS Management Console 内の EventBridge で、[Rules] (ルール) を選択した後に、名前および説明を入力します。[Default] (デフォルト) イベントバスにルールを作成します。
2. ステップ 2 で、イベントソースに [その他] を選択し、次に [イベントパターン] で以下のソースと一致させます。

```
{
  "source": ["aws.internetmonitor"]
}
```

3. ステップ 3 で、ターゲットに [AWS のサービス] と [CloudWatch Logs グループ] を選択してから、既存のロググループを選択するか、新しいロググループを作成します。
4. 必要なタグを追加して、ルールを作成します。これにより、選択した CloudWatch Logs グループに対し EventBridge からイベントが入力されます。

EventBridge ルールでイベントパターンがどのように機能するかについては、「Amazon EventBridge ユーザーガイド」の「Amazon EventBridge のイベントパターン」を参照してください。

CloudWatch ログとメトリクスのアクセスエラーのトラブルシューティング

サポートする機能によっては、Amazon CloudWatch Internet Monitor からログやメトリクスといった特定の Amazon CloudWatch リソースを操作する必要があります。必要な CloudWatch リソースにアクセスしようとしてもアクセスできない場合、Internet Monitor はモニタのステータスコードを FAULT_ACCESS_CLOUDWATCH に設定します。

モニタが FAULT_ACCESS_CLOUDWATCH 状態になる理由はいくつかあります。以下のセクションでは、これらのエラーの考えられる原因と、推奨されるトラブルシューティングの手順を示します。

Internet Monitor から自分のアカウントの CloudWatch ログにアクセスできない

Internet Monitor には、追跡しているアプリケーショントラフィックに関する診断ログが公開されます。こうしたログは、CloudWatch Logs 内の ロググループに公開されます。場所は `/aws/internet-monitor/monitor_name/[byCity|byMetro|bySubdivision|byCountry]` です。Internet Monitor がこれらのロググループにアクセスできませんでした。

エラーの状態と考えられる解決策:

- PutLogEvents スロットリングエラー: Internet Monitor サービスが、そのログを CloudWatch に公開しようとしたときにスロットルされた可能性があります。アカウントのスロットリング制限を確認し、必要に応じて制限の引き上げをリクエストしてください。
- ロググループは存在しません: モニタをいったん無効にしてから再度有効にします。モニタを有効にするとロググループの作成が再開され、問題が解決する可能性があります。
- PutLogEvents アクセス拒否エラー: AWS サポートにお問い合わせください。
- PutLogEvents の不明または一般的なエラー: AWS サポートにお問い合わせください。

Internet Monitor から自分のアカウントの CloudWatch メトリクスにアクセスできない

Internet Monitor は、モニタによって追跡されるアプリケーショントラフィックについて特定の CloudWatch メトリクスを提供します。Internet Monitor がこうしたメトリクスを CloudWatch に提供しようとしたときに、エラーが発生しました。

エラーの状態と考えられる解決策:

- PutMetricData スロットリングエラー: Internet Monitor サービスが、そのメトリクスを CloudWatch に公開しようとしたときにスロットルされた可能性があります。アカウントのスロットリング制限を確認し、必要に応じて制限の引き上げをリクエストしてください。

- PutMetricData アクセス拒否エラー: AWS サポートにお問い合わせください。
- PutMetricData の不明または一般的なエラー: AWS サポートにお問い合わせください。

Amazon CloudWatch Internet Monitor でのデータ保護とデータプライバシー

Amazon CloudWatch Internet Monitor でのデータ保護とデータプライバシーには、AWS の責任共有モデルが適用されます。このモデルで定義されるように、AWS は、すべての AWS クラウドを実行するグローバルなインフラストラクチャの保護に責任を負います。お客様は、このインフラストラクチャでホストされているコンテンツに対する管理を維持する責任があります。データプライバシーの詳細については、「データプライバシーのよくある質問」を参照してください。欧州でのデータ保護の詳細については、AWS セキュリティブログの記事「The AWS Shared Responsibility Model and GDPR」を参照してください。GDPR 要件の遵守に関するその他のリソースについては、一般データ保護規則 (GDPR) センターを参照してください。

エンドユーザーのアカウント番号、E メールアドレス、その他の個人情報などの機密性がある識別情報は、自由形式のフィールドに格納することは避けるように強くお勧めします。Amazon CloudWatch Internet Monitor、または他のサービスに入力したデータは、診断ログに含まれる可能性があります。

Amazon CloudWatch Internet Monitor 用 Identity and Access Management

AWS Identity and Access Management (IAM) は、管理者が AWS リソースへのアクセスを安全に制御するために役立つ AWS のサービスです。IAM の管理者は、誰を認証 (サインインを許可) し、誰に Internet Monitor リソースの使用を承認する (アクセス許可を持たせる) かを制御します。IAM は、追加費用なしで使用できる AWS のサービス です。

> **Important**
>
> Internet Monitor リソースの変更 (2023 年 2 月 24 日)
> 2023 年 2 月 24 日より前に Internet Monitor リソースを含む IAM ポリシーを作成した場合は、Internet Monitor のリソースとリソースタイプが次のように変更されていることに注意してください。
>
> - HealthEvents リソースの名前が HealthEvent に変更されました。
> - HealthEvent リソースの ARN と正規表現の形式が更新されました。

- Monitor リソースの ARN と正規表現の形式が更新されました。
- GetHealthEvent アクションのリソースレベルのアクセス許可は、HealthEvent リソースタイプでのみサポートされるようになりました。Monitor リソースではサポートされていません。
- Monitor リソースタイプの TagResource、UntagResource、ListTagsForResource が必須になるように更新されました。

Internet Monitor 内の AWS リソースへのアクセスを管理するポリシーで指定できるアクション、リソース、条件キーの詳細については、「Amazon CloudWatch Internet Monitor のアクション、リソース、および条件キー」を参照してください。

内容

IAM と Amazon CloudWatch Internet Monitor の連携

IAM を使用して Internet Monitor へのアクセスを管理する前に、Internet Monitor で利用できる IAM 機能について学びましょう。

AWS のサービスが大部分の IAM 機能と連動する方法に関する同様の概要のビューを表示するテーブルを確認するには、「IAM ユーザーガイド」の「IAM と連携する AWS のサービス」を参照してください。

Amazon CloudWatch Internet Monitor で利用できる IAM 機能

IAM の機能	Internet Monitor のサポート
アイデンティティベースのポリシー	Yes
リソースベースのポリシー	No

IAM の機能	Internet Monitor のサポート
ポリシーアクション	Yes
ポリシーリソース	はい
ポリシー条件キー (サービス固有)	はい
ACL	No
ABAC (ポリシー内のタグ)	部分的
一時的な認証情報	Yes
プリンシパル権限	Yes
サービスロール	いいえ
サービスリンクロール	はい

Internet Monitor のアイデンティティベースのポリシー

アイデンティティベースポリシーをサポートする	Yes

アイデンティティベースポリシーは、IAM ユーザー、ユーザーグループ、ロールなど、アイデンティティにアタッチできる JSON 権限ポリシードキュメントです。これらのポリシーは、ユーザーとロールが実行できるアクション、リソース、および条件をコントロールします。アイデンティティベースのポリシーを作成する方法については、『IAM ユーザーガイド』の「IAM ポリシーの作成」を参照してください。

IAM アイデンティティベースのポリシーでは、許可または拒否するアクションとリソース、およびアクションを許可または拒否する条件を指定できます。プリンシパルは、それが添付されているユーザーまたはロールに適用されるため、アイデンティティベースのポリシーでは指定できません。JSON ポリシーで使用できるすべての要素については、「IAM ユーザーガイド」の「IAM JSON ポリシーの要素のリファレンス」を参照してください。

Internet Monitor 内のリソースベースのポリシー

リソースベースのポリシーのサポート	No

リソースベースのポリシーは、リソースに添付する JSON ポリシードキュメントです。リソースベースのポリシーには例として、IAM ロールの信頼ポリシーや Amazon S3 バケットポリシーがあげられます。リソースベースのポリシーをサポートするサービスでは、サービス管理者はポリシーを使用して特定のリソースへのアクセスを制御できます。

Internet Monitor のポリシーアクション

ポリシーアクションに対するサポート	Yes

管理者は AWS JSON ポリシーを使用して、誰が何にアクセスできるかを指定できます。つまり、どのプリンシパルがどんなリソースにどんな条件でアクションを実行できるかということです。

JSON ポリシーの`Action`要素には、ポリシー内のアクセスを許可または拒否するために使用できるアクションが記述されます。ポリシーアクションの名前は通常、関連する AWSAPI オペレーションと同じです。一致する API オペレーションのない権限のみのアクションなど、いくつかの例外があります。また、ポリシーに複数アクションが必要なオペレーションもあります。これらの追加アクションは、 依存アクション と呼ばれます。

このアクションは、関連付けられたオペレーションを実行するための権限を付与するポリシーで使用されます。

Internet Monitor アクションのリストを確認するには、「サービス認証リファレンス」の「Amazon CloudWatch Internet Monitor で定義されるアクション」を参照してください。

Internet Monitor のポリシーアクションは、アクションの前に以下のプレフィックスを使用します。

```
internetmonitor
```

単一のステートメントで複数のアクションを指定するには、アクションをカンマで区切ります。

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

```
        ]
```

ワイルドカード (*) を使用して複数アクションを指定できます。例えば、Describe という単語で始まるすべてのアクションを指定するには、次のアクションを含めます。

```
"Action": "internetmonitor:Describe*"
```

Internet Monitor のポリシーリソース

ポリシーリソースに対するサポート	はい

「サービス認証リファレンス」では、Internet Monitor に関連する次の情報を確認できます。

- Internet Monitor リソースタイプおよびその ARN のリストを表示するには、「Amazon CloudWatch Internet Monitor で定義されるリソース」を参照してください。
- どのアクションで各リソースの ARN を指定できるかについては、「 Amazon CloudWatch Internet Monitor で定義されるアクション」を参照してください。

管理者は AWS JSON ポリシーを使用して、誰が何にアクセスできるかを指定できます。つまり、どのプリンシパルがどのリソースにどのような条件でアクションを実行できるかということです。

Resource JSON ポリシー要素は、アクションが適用されるオブジェクトを指定します。ステートメントには、Resource または NotResource要素を含める必要があります。ベストプラクティスとして、Amazon リソースネーム (ARN) を使用してリソースを指定します。これは、リソースレベルの権限と呼ばれる特定のリソースタイプをサポートするアクションに対して実行できます。

オペレーションのリスト化など、リソースレベルの権限をサポートしないアクションの場合は、ステートメントがすべてのリソースに適用されることを示すために、ワイルドカード (*) を使用します。

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

Internet Monitor のポリシー条件キー

サービス固有のポリシー条件キーのサポート	はい

管理者は AWS JSON ポリシーを使用して、誰が何にアクセスできるかを指定できます。つまり、どのプリンシパルがどんなリソースにどんな条件でアクションを実行できるかということです。

Condition 要素 (または Condition ブロック) を使用すると、ステートメントが有効な条件を指定できます。Condition 要素はオプションです。イコールや未満などの 条件演算子 を使用して条件式を作成することで、ポリシーの条件とリクエスト内の値を一致させることができます。

1 つのステートメントに複数の Condition 要素を指定するか、1 つの Condition 要素に複数のキーを指定すると、AWS は AND 論理演算子を使用してそれらを評価します。単一の条件キーに複数の値を指定すると、AWS は OR 論理演算子を使用して条件を評価します。ステートメントの権限が付与される前にすべての条件が満たされる必要があります。

条件を指定する際にプレースホルダー変数も使用できます。例えば IAM ユーザーに、IAM ユーザー名がタグ付けされている場合のみリソースにアクセスできる権限を付与することができます。詳細については、『IAM ユーザーガイド』の「IAM ポリシーの要素: 変数およびタグ」を参照してください。

AWS はグローバル条件キーとサービス固有の条件キーをサポートしています。すべての AWS グローバル条件キーを確認するには、IAM ユーザーガイド の「AWS グローバル条件コンテキストキー」を参照してください。

Internet Monitor の条件キーのリストを確認するには、「サービス認証リファレンス」の「Amazon CloudWatch Internet Monitor の条件キー」を参照してください。どのアクションやリソースで条件キーを使用できるかについては、「Amazon CloudWatch Internet Monitor で定義されるアクション」を参照してください。

Internet Monitor のACL

ACL のサポート	No

アクセスコントロールリスト (ACL) は、どのプリンシパル (アカウントメンバー、ユーザー、またはロール) がリソースにアクセスするための権限を持つかを制御します。ACL はリソースベースのポリシーに似ていますが、JSON ポリシードキュメント形式は使用しません。

Internet Monitor での ABAC

ABAC (ポリシー内のタグ) のサポート	部分的

Internet Monitor は、ポリシー内のタグの一部をサポートしています。リソースの 1 つであるモニタのタグ付けをサポートしています。

Internet Monitor でタグを使用するには、AWS Command Line Interface または AWS SDK を使用します。AWS Management Console では、Internet Monitor のタグ付けをサポートしていません。

ポリシーにおけるタグの使用全般の詳細については、次の情報を確認してください。

属性ベースのアクセス制御 (ABAC) は、属性に基づいて権限を定義する認可戦略です。AWS では、属性は タグ と呼ばれます。タグは、IAM エンティティ (ユーザーまたはロール)、および多数の AWS リソースにアタッチできます。エンティティとリソースのタグ付けは、ABAC の最初の手順です。その後、プリンシパルのタグがアクセスしようとしているリソースのタグと一致した場合に操作を許可するように ABAC ポリシーを設計します。

ABAC は、急成長する環境やポリシー管理が煩雑になる状況で役立ちます。

タグに基づいてアクセスを管理するには、`aws:ResourceTag/key-name`、`aws:RequestTag/key-name`、または `aws:TagKeys` の条件キーを使用して、ポリシーの条件要素でタグ情報を提供します。

サービスがすべてのリソースタイプに対して 3 つの条件キーすべてをサポートする場合、そのサービスの値ははいです。サービスが一部のリソースタイプに対してのみ 3 つの条件キーのすべてをサポートする場合、値は「部分的」になります。

ABAC の詳細については、『IAM ユーザーガイド』の「ABAC とは?」を参照してください。ABAC をセットアップするステップを説明するチュートリアルについては、「IAM ユーザーガイド」の「属性に基づくアクセスコントロール (ABAC) を使用する」を参照してください。

Internet Monitor で一時的な認証情報を使用する

一時的な認証情報のサポート	Yes

AWS のサービスには、一時認証情報を使用してサインインしても機能しないものがあります。一時的な認証情報を利用できる AWS のサービス を含めた詳細情報については、『IAM ユーザーガイド』の「IAM と連携する AWS のサービス」を参照してください。

ユーザー名とパスワード以外の方法で AWS Management Consoleにサインインする場合は、一時認証情報を使用していることになります。例えば、会社のシングルサインオン (SSO) リンクを使用して AWS にアクセスすると、そのプロセスは自動的に一時認証情報を作成します。また、ユーザーと

してコンソールにサインインしてからロールを切り替える場合も、一時的な認証情報が自動的に作成されます。ロールの切り替えに関する詳細については、「IAM ユーザーガイド」の「ロールへの切り替え (コンソール)」を参照してください。

一時認証情報は、AWS CLI または AWSAPI を使用して手動で作成できます。作成後、一時認証情報を使用して AWSにアクセスできるようになります。AWS は、長期的なアクセスキーを使用する代わりに、一時認証情報を動的に生成することをお勧めします。詳細については、「IAM の一時的セキュリティ認証情報」を参照してください。

Internet Monitor のクロスサービスプリンシパル許可

フォワードアクセスセッション (FAS) をサポート	Yes

IAM ユーザーまたはロールを使用して AWS でアクションを実行するユーザーは、プリンシパルとみなされます。一部のサービスを使用する際に、アクションを実行してから、別のサービスの別のアクションを開始することがあります。FAS は、AWS のサービスを呼び出すプリンシパルの権限を、AWS のサービスのリクエストと合わせて使用し、ダウンストリームのサービスに対してリクエストを行います。FAS リクエストは、サービスが、完了するために他の AWS のサービス またはリソースとのやりとりを必要とするリクエストを受け取ったときにのみ行われます。この場合、両方のアクションを実行するためのアクセス許可が必要です。FAS リクエストを行う際のポリシーの詳細については、「転送アクセスセッション」を参照してください。

Internet Monitor のサービスロール

サービスロールのサポート	いいえ

サービスロールとは、サービスがユーザーに代わってアクションを実行するために引き受ける IAM ロールです。IAM 管理者は、IAM 内からサービスロールを作成、変更、削除できます。詳細については、「IAM ユーザーガイド」の「AWS のサービス に権限を委任するロールの作成」を参照してください。

Internet Monitor のサービスリンクロール

サービスリンクロールのサポート	Yes

サービスリンクロールは、AWS のサービス にリンクされているサービスロールの一種です。サービスがロールを引き受け、ユーザーに代わってアクションを実行できるようになります。サービスリンクロールは、AWS アカウント に表示され、サービスによって所有されます。IAM 管理者は、サービスにリンクされたロールの権限を表示できますが、編集することはできません。

Internet Monitor のサービスリンクロールの詳細については、「Amazon CloudWatch Internet Monitor のサービスリンクロール」を参照してください。

AWS のサービスリンクロール全般の作成または管理の詳細については、「IAM と提携する AWS のサービス」を参照してください。表の中から、[Service-linked role] (サービスにリンクされたロール) 列に Yes と記載されたサービスを見つけます。サービスにリンクされたロールに関するドキュメントをサービスで表示するには、[Yes] リンクを選択します。

Amazon CloudWatch Internet Monitor の AWS マネージドポリシー

AWS マネージドポリシーは、AWS が作成および管理するスタンドアロンポリシーです。AWS マネージドポリシーは、多くの一般的なユースケースで権限を提供できるように設計されているため、ユーザー、グループ、ロールへの権限の割り当てを開始できます。

AWS マネージドポリシーは、ご利用の特定のユースケースに対して最小特権の権限を付与しない場合があることにご注意ください。AWS のすべてのお客様が使用できるようになるのを避けるためです。ユースケース別にカスタマーマネージドポリシーを定義して、マネージドポリシーを絞り込むことをお勧めします。

AWS マネージドポリシーで定義したアクセス権限は変更できません。AWS が AWS マネージドポリシーに定義されている権限を更新すると、更新はポリシーがアタッチされているすべてのプリンシパルアイデンティティ (ユーザー、グループ、ロール) に影響します。新しい AWS のサービス を起動するか、既存のサービスで新しい API オペレーションが使用可能になると、AWS が AWS マネージドポリシーを更新する可能性が最も高くなります。

詳細については、「IAM ユーザーガイド」の「AWS マネージドポリシー」を参照してください。

AWS マネージドポリシー: CloudWatchInternetMonitorServiceRolePolicy

このポリシーは、AWSServiceRoleForInternetMonitor という名前のサービスリンクロールにアタッチされます。このロールにより、Internet Monitor は Amazon Virtual Private Cloud リソースや Network Load Balancer などのアカウント内リソースにアクセスできるようになるため、モニター作成時にそれらのリソースを選択できます。詳細については、「Amazon CloudWatch Internet Monitor のサービスリンクロール」を参照してください。

Amazon CloudWatch Internet Monitor 用の IAM 許可

Amazon CloudWatch Internet Monitor でモニタとデータを操作できるように必要なアクションにアクセスするには、ユーザーに適切なアクセス許可が必要です。

Amazon CloudWatch のセキュリティの詳細については、「Amazon CloudWatch 用 Identity and Access Management」を参照してください。

Amazon CloudWatch Internet Monitor の読み取り専用アクセス許可

Amazon CloudWatch Internet Monitor でモニタとデータを操作できるように読み取り専用アクションにアクセスするには、以下のアクセス許可が付与されたユーザーまたはロールとしてサインインする必要があります。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "cloudwatch:GetMetricData",
                "internetmonitor:Get*",
                "internetmonitor:List*",
                "internetmonitor:StartQuery",
                "internetmonitor:StopQuery",
                "logs:DescribeLogGroups",
                "logs:GetQueryResults",
                "logs:StartQuery",
                "logs:StopQuery"
                ],
            "Resource": "*"
        }
    ]
}
```

Amazon CloudWatch Internet Monitor のフルアクセス許可

Amazon CloudWatch Internet Monitor でモニタを作成し、Internet Monitor でモニタとデータを操作できるように必要なアクションにフルアクセスするには、以下のアクセス許可が付与されたユーザーまたはロールとしてサインインする必要があります。

- Internet Monitor に関連付けられているサービスリンクロールを作成するためのアクセス許可。詳細については、「Amazon CloudWatch Internet Monitor のサービスリンクロール」を参照してください。
- Internet Monitor でモニタとデータを操作できるように必要なアクションにフルアクセスできるアクセス許可。

> **Note**
>
> より制限の厳しいアイデンティティベースのアクセス許可ポリシーを作成した場合、そのポリシーを持つユーザーには Internet Monitor でモニタを作成してデータを操作するために必要なフルアクセスを付与できない可能性があります。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "internetmonitor:*"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": "iam:CreateServiceLinkedRole",
            "Resource": "arn:aws:iam::*:role/aws-service-role/
internetmonitor.amazonaws.com/AWSServiceRoleForInternetMonitor",
            "Condition": {
                "StringLike": {
                    "iam:AWSServiceName": "internetmonitor.amazonaws.com"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "iam:AttachRolePolicy",
                "iam:PutRolePolicy"
            ],
```

```
            "Resource": "arn:aws:iam::*:role/aws-service-role/
internetmonitor.amazonaws.com/AWSServiceRoleForInternetMonitor"
        },
        {
            "Action": [
                "ec2:DescribeVpcs",
                "elasticloadbalancing:DescribeLoadBalancers",
                "workspaces:DescribeWorkspaceDirectories",
                "cloudfront:GetDistribution"
            ],
            "Effect": "Allow",
            "Resource": "*"
        }
    ]
}
```

Amazon CloudWatch Internet Monitor のサービスリンクロール

Amazon CloudWatch Internet Monitor は AWS Identity and Access Management (IAM) サービスリンクロールを使用します。サービスにリンクされたロールは、Internet Monitor に直接リンクされた一意のタイプの IAM ロールです。サービスリンクロールは Internet Monitor によって事前に定義されており、サービスがユーザーに代わって他の AWS サービスを呼び出すために必要なアクセス許可がすべて含まれています。

Internet Monitor は、サービスにリンクされたロールのアクセス許可を定義します。特に定義されている場合を除き、Internet Monitor のみがそのロールを引き受けることができます。定義したアクセス許可には、信頼ポリシーと許可ポリシーが含まれます。この許可ポリシーを他の IAM エンティティにアタッチすることはできません。

ロールを削除するには、まずそのロールの関連リソースを削除します。この制限により、リソースへのアクセス許可を誤って削除することがなくなるため、Internet Monitor リソースは保護されます。

サービスにリンクされたロールをサポートする他のサービスについては、「IAM と連動する AWS サービス」を参照し、[Service-linked roles] (サービスにリンクされたロール) の列内で [Yes] (はい) と表記されたサービスを確認してください。サービスにリンクされたロールに関するドキュメントをサービスで表示するには、はいリンクを選択します。

Internet Monitor のサービスにリンクされたロールのアクセス許可

Internet Monitor は、AWSServiceRoleForInternetMonitor という名前のサービスリンクロールを使用します。このロールにより、Internet Monitor は Amazon Virtual Private Cloud リソース、Amazon

CloudFront ディストリビューション、Amazon WorkSpaces ディレクトリ、Network Load Balancers といったアカウント内のリソースにアクセスできます。そして、モニタを作成するときにこれらのリソースを選択できます。

このサービスリンクロールは、マネージドポリシーである `CloudWatchInternetMonitorServiceRolePolicy` を使用します。

AWSServiceRoleForInternetMonitor サービスリンクロールは、次のサービスを信頼してロールを引き受けます。

- `internetmonitor.amazonaws.com`

このポリシーに対する許可を確認するには、「AWS マネージドポリシーリファレンス」の「CloudWatchInternetMonitorServiceRolePolicy」を参照してください。

Internet Monitor のサービスにリンクされたロールの作成

Internet Monitor のサービスにリンクされたロールを手動で作成する必要はありません。初めてモニターを作成すると、Internet Monitor によって AWSServiceRoleForInternetMonitor が自動的に作成されます。

詳細については、『IAM ユーザーガイド』の「サービスにリンクされたロールの作成」を参照してください。

Internet Monitor のサービスにリンクされたロールの編集

Internet Monitor がサービスにリンクされたロールをアカウントに作成すると、多くのエンティティによってロールが参照される可能性があるため、ロールの名前を変更することはできません。IAM を使用してロールの説明を編集することはできます。詳細については、「IAM ユーザーガイド」の「サービスリンクロールの編集」を参照してください。

Internet Monitor のサービスにリンクされたロールの削除

サービスにリンクされたロールが必要な機能またはサービスが不要になった場合には、そのロールを削除することをお勧めします。そうすることで、モニタリングや保守が積極的に行われていない未使用のエンティティを排除できます。ただし、手動で削除する前に、サービスにリンクされたロールのリソースをクリーンアップする必要があります。

Internet Monitor のモニターからリソースを削除した後にモニターを削除すると、サービスにリンクされたロール AWSServiceRoleForInternetMonitor を削除できます。

> **Note**
>
> リソースを削除する際に、Internet Monitor サービスでロールが使用されている場合、削除は失敗することがあります。その場合は、数分待ってから再試行してください。

サービスにリンクされたロールを IAM で手動削除するには

IAM コンソール、AWS CLI、または AWS API を使用して、AWSServiceRoleforInternetMonitor サービスリンクロールを削除します。詳細については、 IAM ユーザーガイド の「サービスにリンクされたロールの削除」を参照してください。

Internet Monitor のサービスリンクロールを更新する

Internet Monitor のサービスリンクロールの AWS マネージドポリシーである AWSServiceRoleForInternetMonitor の更新については、「AWS マネージドポリシーに関する CloudWatch 更新」を参照してください。CloudWatch のマネージドポリシーの変更に関する自動通知を入手するには、CloudWatch ドキュメントの履歴ページから、RSS フィードにサブスクライブしてください。

Amazon CloudWatch Internet Monitor のクォータ

Amazon CloudWatch Internet Monitor には、次のクォータがあります。

リソース	デフォルトのクォータ
リージョンあたりのモニター数	50
モニターあたりのリソース数	50
Internet Monitor で解決されたヘルスイベントが保持される日数	400

Amazon CloudWatch Network Monitor の使用

Amazon CloudWatch Network Monitor は、AWS ホストアプリケーションをオンプレミスの宛先に接続するネットワークのパフォーマンスを可視化し、ネットワークパフォーマンスの低下の原因を数

分以内に特定できるようにします。Network Monitor は AWS による完全マネージド型のサービスです。そのため、ネットワークパフォーマンスをモニタリングするために追加のエージェントをインストールする必要はありません。ハイブリッドネットワーク接続のパケット損失とレイテンシーを迅速に視覚化し、アラートとしきい値を設定して、エンドユーザーのネットワークエクスペリエンスを向上させるための対策を講じることができます。

Network Monitor は、ネットワークのパフォーマンスをリアルタイムで把握しようとするネットワークオペレーターとアプリケーション開発者を対象としています。

主な特徴

- Network Monitor を使用すると、継続的にリアルタイムのパケット損失とレイテンシーのメトリクスを使って、変化するハイブリッドネットワーク環境のベンチマークを行うことができます。
- AWS Direct Connect を使用して接続すると、Network Monitor は CloudWatch アカウントに AWS ネットワークヘルスインジケータを書き込み、ネットワークのパフォーマンス低下を迅速に診断します。このメトリクスは、ネットワークのパフォーマンス低下が AWS 内で発生したかどうかを判断するための確率的スコアを提供します。
- Network Monitor は、フルマネージド型エージェントを使用してスムーズなモニタリングを実現します。つまり、VPC にもオンプレミスにもエージェントをインストールする必要はありません。VPC サブネットとオンプレミス IP アドレスを指定するだけで使用を開始できます。
- Network Monitor は CloudWatch メトリクスにメトリクスを公開します。ダッシュボードを作成して、メトリクスを表示したり、アプリケーションに固有のメトリクスに基づく実用的なしきい値やアラームを作成したりできます。

詳細については、「the section called “Network Monitor の仕組み”」を参照してください。

Network Monitor の用語とコンポーネント

- モニタ — モニタには、ネットワークのパフォーマンスと可用性の測定値を表示したいリソースや、ヘルスイベントアラートの受け取りが必要なリソースが表示されます。アプリケーションのモニタを作成するときは、ネットワークソースとして AWS ホストリソースを追加します。Network Monitor はその後、AWS ホストリソースと宛先 IP アドレスの間で組み合わせ可能なすべてのプローブのリストを作成します。
- プローブ — プローブは、AWS ホストリソースからオンプレミスの宛先 IP アドレスに送信されるトラフィックです。モニタに設定した各のプローブの Network Monitor メトリクスが CloudWatch アカウントに書き込まれます。

- AWS ネットワークソース — これはネットワークモニタプローブの送信元の AWS ソースで、いずれかの VPC 内のサブネットになります。
- 宛先 — オンプレミスネットワーク内にある、AWS ネットワークソースのターゲットです。宛先は、オンプレミスの IP アドレス、ネットワークプロトコル、ポート、およびネットワークパケットサイズで構成されます。IPv4 と IPv6 両方のアドレスがサポートされます。

Network Monitor の制限事項と要件

- Network Monitor は、最大 4 つの宛先 IP アドレスと、モニタあたり最大 24 個のプローブをサポートします。
- アカウントごとの各リージョンで、最大 100 個のモニタを使用できます。
- モニターサブネットは、モニターと同じアカウントが所有している必要があります。
- Network Monitor は、AWS ネットワークに問題が発生した場合に、自動ネットワークフェイルオーバーを提供しません。
- 作成するプローブごとに料金が発生します。料金の詳細については、「the section called “料金”」を参照してください。

Amazon CloudWatch Network Monitor の仕組み

Network Monitor は、フルマネージド型のエージェントレスソリューションを提供することで、容易なモニタリングを実現します。AWS ホストリソースにモニタを作成すると、AWS はすべてのインフラストラクチャをバックグラウンドで作成して管理し、ラウンドトリップタイムとパケット損失の測定を行います。そのため、AWS インフラストラクチャ内へのエージェントのインストールやアンインストールを必要とせずに、モニタリングを迅速に拡張できます。

Network Monitor は、AWS リージョン からのすべてのフローを広範囲にモニタリングするのではなく、AWS ホストリソースからのフローがたどるルートを重点的にモニタリングします。ワークロードが複数のアベイラビリティーゾーン (AZ) に分散している場合、Network Monitor は各プライベートサブネットからのルートをモニタリングできます。

Network Monitor は、モニタの作成時に設定された集計間隔に基づいて、ラウンドトリップタイムとパケット損失のメトリクスを Amazon CloudWatch アカウントに公開します。CloudWatch を使用して、モニタごとに個別のレイテンシーとパケット損失のしきい値を設定することもできます。例えば、パケット損失の影響を受けやすいワークロードのパケット損失平均が静的 0.1% のしきい値を超えた場合にユーザーに通知するアラームを作成できます。また、CloudWatch の異常検出機能を使用

して、パケット損失やレイテンシーのメトリクスが望ましい範囲を超えたときにアラームを出すこともできます。

可用性とパフォーマンスの測定

Network Monitor は、AWS リソースからオンプレミスの宛先に定期的にアクティブなプローブを送信します。モニタを作成する際には、以下を指定できます。

- 集計間隔。CloudWatch が測定結果を受け取る時間 (秒単位)。これは 30 秒ごと、または 60 秒ごとになります。モニタに選択した集計間隔は、そのモニタ内のすべてのプローブに適用されます。
- プローププロトコル。モニタに追加された各プローブは、Internet Control Message Protocol (ICMP) または Transmission Control Protocol (TCP) のいずれかのプロトコルを使用する必要があります。詳細については、「the section called “通信プロトコル”」を参照してください。
- パケットサイズ。1 つのプローブで AWS ホストリソースと宛先との間で送信される各パケットのサイズ (バイト単位)。モニタ内の各プローブには独自のパケットサイズを設定できます。

メトリクス:

- ラウンドトリップタイムメトリクス (ミリ秒単位で測定) は、パフォーマンスの測定値を測定して記録します。プローブが宛先 IP アドレスに送信されるまでの時間と、関連する応答が受信されるまでの時間を記録します。
- パケット損失メトリクスは、送信された総パケットの割合を測定し、関連する応答を受信しなかった送信済みプローブの数を記録します。これは、これらのパケットがネットワークパス上で事実上失われたことを意味します。

サポートされる通信プロトコル

ICMP ベースのプローブは、AWS ホストリソースからの ICMP エコー要求を宛先アドレスに送信し、宛先アドレスから ICMP エコー応答が返されることを期待します。Network Monitor は、ICMP エコー要求と応答メッセージの情報を使用して、ラウンドトリップタイムとパケット損失のメトリクスを計算します。

TCP ベースのプローブは、AWS ホストリソースからの TCP SYN パケットを宛先のアドレスとポートに送信し、宛先のアドレスとポートから TCP SYN+ACK または RST パケットが返されることを期待します。Network Monitor は、TCP SYN および TCP SYN+ACK または RST メッセージの情報を使用して、ラウンドトリップタイムとパケット損失のメトリクスを計算します。さらに、Network

Monitor はソース TCP ポートを定期的に切り替えてネットワークカバレッジを広げます。これにより、パケット損失を検出できる可能性を高めています。

AWS ネットワークヘルスインジケータ

Network Monitor は、AWS Direct Connect を介して接続されている宛先のネットワークパフォーマンスと可用性に関する情報を提供するネットワークヘルスインジケータ (NHI) メトリクスを公開します。このメトリクスは、モニタがデプロイされている AWS ホストリソースから Direct Connect ロケーションまでの、AWS によって制御されているネットワークパスの状態を示す統計的測定値です。

Network Monitor は、異常検出機能を使用して、ネットワークパスにおける可用性の低下やパフォーマンスの低下を計算します。

> **Note**
>
> 新しいモニタを作成したり、プローブを追加したり、プローブを再度有効にしたりするたびに、そのモニタの NHI に数時間遅延が生じます。その間に AWS はデータを収集して異常検出を実行します。

NHI ヘルスメトリクスを提供するために、Network Monitor は、AWS サンプルデータセット全体にも、ネットワークパスをシミュレートするトラフィックのパケット損失メトリクスと往復遅延メトリクスにも統計的相関も適用します。このメトリクスは、1 または 0 の 2 つの変数のいずれかになります。値が 1 の場合は、Network Monitor が、AWS によって制御されているネットワークパス内でネットワークパフォーマンスの低下を観察したことを示します。値が 0 の場合は、Network Monitor が、このパスでネットワークパフォーマンスの低下を一切観察しなかったことを示します。このメトリクスにより、ネットワークの問題をより迅速にトラブルシューティングできます。NHI メトリクスにアラートを設定すると、ネットワークパスで発生している問題について通知を受けることができます。

IPv4 アドレスと IPv6 アドレスのサポート

Network Monitor は、IPv4 または IPv6 ネットワーク上の可用性とパフォーマンスのメトリクスを提供し、デュアルスタック VPC の IPv4 または IPv6 アドレスをモニタリングできます。Network Monitor では、同じモニタ内に IPv4 と IPv6 両方の宛先を設定することはできませんが、IPv4 のみの宛先と IPv6 のみの宛先を別々に作成することができます。

利用可能なリージョン

Network Monitor を現在利用できるのは、次の AWS リージョンです。

リージョン	
アジアパシフィック (香港)	ap-east-1
アジアパシフィック (ムンバイ)	ap-south-1
アジアパシフィック (ソウル)	ap-northeast-2
アジアパシフィック (シンガポール)	ap-southeast-1
アジアパシフィック (シドニー)	ap-southeast-2
アジアパシフィック (東京)	ap-northeast-1
カナダ西部 (カルガリー)	ca-west-1

リージョン	
欧州 (フランクフルト)	eu-central-1
欧州 (アイルランド)	eu-west-1
欧州 (ロンドン)	eu-west-2
欧州 (パリ)	eu-west-3
欧州 (ストックホルム)	eu-north-1
中東 (バーレーン)	me-south-1
南米 (サンパウロ)	sa-east-1
米国東部 (バージニア北部)	us-east-1
米国東部 (オハイオ)	us-east-2

リージョン	
米国西部 (北カリフォルニア)	us-west-1
米国西部 (オレゴン)	us-west-2

ネットワークモニタの作成

以下の手順では、モニタを作成し、必要なプローブを追加する方法について説明します。プローブには、ソースサブネットと最大 4 つの宛先 IP アドレスを選択します。モニタあたりの最大プローブ数は 24 個です。モニタを作成するには、Amazon CloudWatch コンソールを使用するか、コマンドラインまたは API を使用します。

トピック

- コンソールを使用したネットワークモニタの作成
- コマンドラインまたは API を使用したネットワークモニタの作成

コンソールを使用したネットワークモニタの作成

次の手順では、Amazon CloudWatch コンソールを使用してモニタを作成する手順について説明します。ソースサブネットを選択したら、最大 4 つの宛先を追加します。モニタあたり最大 24 個のプローブを作成できます。モニタを作成するには、Amazon CloudWatch コンソールを使用するか、コマンドラインまたは SDK を使用します。

> **Important**
>
> これらの手順は、すべてを一度に完了することを想定しています。実行中の作業を保存して、後で続行することはできません。

モニタの詳細の定義

モニタを作成する最初の手順は、基本的な詳細の定義です。これには、モニタ名の設定や集計期間の定義も含まれます。必要に応じて、モニタにタグを追加することができます。

モニタの詳細を定義するには

1. https://console.aws.amazon.com/cloudwatch/ で CloudWatch コンソールを開き、[ネットワークモニタリング] の下で [Network Monitor] を選択します。
2. [Create monitor] (モニターの作成) を選択します。
3. [モニタ名] には、このモニタに使用する名前を入力します。
4. [集計期間] には、CloudWatch にメトリクスを送信する頻度を選択します。利用可能な集計期間は以下のとおりです。

   - 30 秒
   - 60 秒

   > **Note**
   >
   > 集計期間が短いほどネットワークの問題をより早く検出できますが、選択した集計期間は請求体系に影響することがあります。料金の詳細については、「Amazon CloudWatch 料金表」を参照してください。

5. (オプション) このリソースの特定を容易にするために、[タグ] セクションに [キー] と [値] のペアを追加します。これにより、特定の情報で検索またはフィルタリングできるようになります。

   1. [新しいタグを追加] をクリックします。
   2. [キー] の名前と関連する [値] を入力します。
   3. [新しいタグを追加] を選択し、その新しいタグを追加します。

      [新しいタグを追加] を選択して複数のタグを追加することも、[削除] を選択して任意のタグを削除することもできます。

   4. タグをモニタと関連付ける場合は、[モニタによって作成されたプローブにタグを追加] をオンのままにします。これによりタグがモニタプローブに追加され、タグベースの認証や計測を使用する場合に役立ちます。

6. [次へ] を選択して「the section called “ソースと宛先の選択”」に進みます。

ソースと宛先の選択

ネットワークモニタは、ネットワークを運用しているリージョンの VPC と関連するサブネットの AWS ソースを使用します。モニタの宛先は、オンプレミスの IP アドレス、ネットワークプロトコル、ポート、ネットワークパケットサイズで構成されます。

ソースと宛先の組み合わせをプローブと呼びます。サブネットあたり最大 4 つのプローブ、モニタあたり最大 24 個のプローブを使用できます。

> **Important**
>
> これらの手順は、すべてを一度に完了することを想定しています。実行中の作業を保存して、後で続行することはできません。

ソースと宛先を選択するには

1. [AWS ネットワークソース] で、モニタに含めるサブネットを 1 つ以上選択します。VPC を 1 つ選択することも (その VPC 内のすべてのサブネットが選択されます)、特定のサブネットを選択することもできます。選択する VPC およびサブネットがネットワークモニタのソースになります。
2. [送信先 1] には、オンプレミスネットワークの宛先 IP アドレスを入力します。IPv4 と IPv6 両方のアドレスがサポートされます。
3. [詳細設定] を選択します。
4. このお客様が管理する宛先のネットワークの [プロトコル] を選択します。次のいずれかを選択できます。
   - ICMP
   - TCP
5. [プロトコル] を [TCP] にする場合は、次の情報を入力します。それ以外の場合は、次の手順に進みます。
   1. ネットワークが接続に使用する [ポート] を入力します。ポートは 1 ~ 65535 の数字でなければなりません。
   2. [パケットサイズ] を入力します。これは、ソースと宛先の間のプローブで送信される各パケットのサイズ (バイト単位) です。パケットサイズは 56 ~ 8500 の数値でなければなりません。

6. このモニタに別のオンプレミス宛先を追加するには、[宛先を追加] を選択します。追加する宛先ごとに、この手順を繰り返します。
7. 終了したら [次へ] を選択してプローブの確認に進みます。

プローブの確認

プローブの確認に進むと、モニタのネットワークプローブの組み合わせを確認することができます。このページには、選択したソースと宛先との間で可能な組み合わせがすべて表示されます。例えば、ソースサブネットが 6 つ、宛先 IP が 4 つある場合、組み合わせ可能なプローブは合計 24 通りになります。

> **Important**
>
> - これらの手順は、すべてを一度に完了することを想定しています。実行中の作業を保存して、後で続行することはできません。
> - [プローブの確認] ページには、プローブが有効かどうかは表示されません。したがって、このページをよく確認し、無効なプローブを削除することをお勧めします。無効なプローブを削除しない場合、そのプローブの料金が請求されることがあります。

モニタプローブを確認するには

1. 前提条件:「the section called “ソースと宛先の選択”」を完了していること。
2. [プローブの確認] ページで、ソースと宛先の組み合わせのリストを確認します。
3. モニタから削除するプローブを 1 つ以上選択し、[削除] を選択します。

> **Note**
>
> 削除を確認するプロンプトは表示されません。削除してしまったプローブは、再度設定する必要があります。[ネットワークモニタ] ページの [ネットワークモニタ] セクションから、プローブをモニタに追加し直すことができます。詳細については、「the section called “プローブをモニタに追加する”」を参照してください。

4. [次へ] を選択してモニタの詳細を確認してから、モニタを作成します

確認と作成

モニタとプローブの作成の最後の手順は、モニタとプローブ両方の詳細の確認です。この時点では、どの情報も変更することができます。モニタの確認と作成が完了し、メトリクスの追跡が始まると、すべてのプローブの料金の請求が開始されます。

> **Important**
>
> - この手順は、モニタとプローブの作成時にすべてを一度に完了することを想定しています。実行中の作業を保存して、後で続行することはできません。
> - いずれかのセクションを編集する場合は、編集している場所からモニタの作成手順を踏む必要があります。ただし、以降の手順を再作成する必要はありません。これらのページには、以前に入力された情報が保持されます。

モニタを確認して作成するには

1. [プローブの確認と作成] ページで、変更したいセクションの [編集] を選択します。
2. そのセクションに変更を加えます。
3. [Next] を選択します。
4. 次のいずれかを実行します。
   - 追加のモニタページに必要な変更を加え、[確認と作成] ページに戻るまで [次へ] を選択します。
   - 変更が必要なページが他にない場合は、[確認と作成] ページに戻るまで [次へ] を選択します。
5. [Create monitor] (モニターの作成) を選択します。

   [ネットワークモニタ] ページの [ネットワークモニタ] セクションには、モニタ作成の現在の状態が表示されます。モニタの作成中は、[状態] は [保留中] になります。[状態] が [アクティブ] に変わったら、モニタダッシュボードにアクセスして CloudWatch メトリクスを表示できます。

   モニタダッシュボードの使用の詳細については、「the section called “Network Monitor ダッシュボード”」を参照してください。

> **Note**
>
> 新しく追加されたネットワークモニタがネットワークメトリクスの収集を開始するまでに数分かかる場合があります。

コマンドラインまたは API を使用したネットワークモニタの作成

コマンドラインまたは API を使用して、ネットワークモニタを表示および作成できます。

コマンドラインまたは API を使用してネットワークモニタを作成するには

1. create-monitor を使用してネットワークモニタを作成します。
2. create-probe を使用してネットワークモニタプローブを作成します。

Network Monitor のモニタとプローブの操作

Amazon CloudWatch コンソールを使用するか、コマンドラインまたは API を使用して、モニタとプローブの次のタスクを実行できます。

トピック:

- モニタの編集
- モニタの削除
- プローブのアクティブ化または非アクティブ化
- プローブをモニタに追加する
- プローブの編集
- プローブの削除
- コマンドラインまたは API を使用したリソースのタグ付けまたはタグの解除

モニタの編集

ネットワークモニタの情報は編集できます。例えば名前を変更したり、新しい集計期間を設定したり、タグを追加または削除したりできます。モニタの情報を変更しても、関連するプローブは変更されません。モニタは、Amazon CloudWatch コンソールを使用するか、コマンドラインまたは API を使用して編集できます。

コンソールを使用したモニタの編集

CloudWatch コンソールを使用してモニタを編集できます。

コンソールを使用してモニタを編集するには

1. https://console.aws.amazon.com/cloudwatch/ で CloudWatch コンソールを開き、[ネットワークモニタリング] の下で [Network Monitor] を選択します。
2. [ネットワークモニタ] セクションで、編集するモニタを選択します。
3. モニタのダッシュボードページで [編集] を選択します。
4. [モニタ名] には、モニタの新しい名前を入力します。
5. [集計期間] には、CloudWatch にメトリクスを送信する頻度を選択します。有効な期間は次のとおりです。

   - 30 秒
   - 60 秒

   > **Note**
   >
   > 集計期間が短いほどネットワークの問題をより早く検出できますが、選択した集計期間は請求体系に影響することがあります。料金の詳細については、「Amazon CloudWatch 料金表」を参照してください。

6. (オプション) このリソースの特定を容易にするために、[タグ] セクションに [キー] と [値] のペアを追加します。これにより、特定の情報で検索またはフィルタリングできるようになります。また、現在のいずれかの [キー] の [値] だけを変更することもできます。

   1. [新しいタグを追加] をクリックします。
   2. [キー] の名前と関連する [値] を入力します。
   3. [新しいタグを追加] を選択し、その新しいタグを追加します。

      [新しいタグを追加] を選択して複数のタグを追加することも、[削除] を選択して任意のタグを削除することもできます。
   4. タグをモニタと関連付ける場合は、[モニタによって作成されたプローブにタグを追加] をオンのままにします。これによりタグがモニタプローブに追加され、タグベースの認証や計測を使用する場合に役立ちます。

7. [Save changes] (変更の保存) をクリックします。

CLI または API を使用したモニタの編集

コマンドラインまたは API を使用して、モニタを表示および編集できます。

コマンドラインまたは API を使用してモニタを編集するには

1. モニタの名前がわからない場合は、list-monitors を使用してモニタのリストを取得します。編集するモニタの名前を書き留めておきます。
2. 前の手順で取得したモニタの名前を使って edit-monitor を使用します。

モニタの削除

モニタを削除する前に、モニタの [状態] に関係なく、そのモニタに関連付けられているすべてのプローブを非アクティブ化または削除する必要があります。モニタを非アクティブ化または削除すると、それらのモニタのプローブの料金を請求されなくなります。削除されたモニタを復元することはできません。モニタは、Amazon CloudWatch コンソールを使用するか、コマンドライン/API を使用して削除できます。

プローブが削除または非アクティブ化された場合でも、CloudWatch はそのメトリクスを 15 日間保持します。

コンソールを使用したモニタの削除

CloudWatch コンソールを使用してモニタを削除できます。

コンソールを使用してモニタを削除するには

1. https://console.aws.amazon.com/cloudwatch/ で CloudWatch コンソールを開き、[ネットワークモニタリング] の下で [Network Monitor] を選択します。
2. [ネットワークモニタ] セクションで、削除するモニタを選択します。
3. [アクション] を選択し、[削除] を選択します。
4. アクティブなプローブがある場合は、非アクティブ化するよう求められます。[プローブを非アクティブ化] を選択します。

> **Note**
>
> [プローブを非アクティブ化] を選択した後に、この操作をキャンセルまたは元に戻すことはできません。ただし、非アクティブ化されたプローブはモニタから削除されませ

ん。後で再度アクティブ化できます。「the section called “プローブのアクティブ化または非アクティブ化”」を参照してください。

5. 確認フィールドに **confirm** を入力し、[削除] を選択します。

コマンドラインまたは API を使用したモニタの削除

コマンドラインまたは API を使用してモニタを削除できます。

コマンドラインまたは API を使用してネットワークモニタを削除するには

1. 削除するモニタの名前が必要になります。名前がわからない場合は、list-monitors を使用してモニタのリストを取得します。削除するモニタの名前を書き留めておきます。
2. そのモニタにプローブが含まれているかどうかを確認します。前の手順で取得したモニタ名を使って get-monitor を使用します。これにより、そのモニタに関連するすべてのプローブのリストが返されます。
3. モニタにプローブが含まれている場合は、まずそれらのプローブを非アクティブに設定するか、削除する必要があります。
   - プローブを非アクティブに設定するには、update-probe を使用し、状態を `INACTIVE` に設定します。
   - プローブを削除するには、delete-probe を使用します。
4. プローブを `INACTIVE` に設定するか削除したら、delete-monitor を使用してモニタを削除します。非アクティブなプローブは削除されません。

プローブのアクティブ化または非アクティブ化

モニタのプローブは、必要に応じてアクティブ化または非アクティブ化することができます。現在使用していないものの、将来再び使用する可能性があるプローブは、非アクティブ化できます。プローブを非アクティブ化すると、設定にもう一度時間を費やす必要がなくなります。非アクティブ化したプローブには課金されません。

モニタのステータスは、Amazon CloudWatch コンソールを使用するか、コマンドラインまたは API を使用して変更できます。

コンソールを使用してプローブをアクティブまたは非アクティブに設定する

CloudWatch コンソールを使用して、プローブをアクティブまたは非アクティブに設定できます。

コンソールを使用してプローブをアクティブまたは非アクティブに設定するには

1. https://console.aws.amazon.com/cloudwatch/ で CloudWatch コンソールを開き、[ネットワークモニタリング] の下で [Network Monitor] を選択します。
2. [モニタの詳細] タブを選択します。
3. [プローブ] セクションで、アクティブ化または非アクティブ化するプローブを選択します。
4. [アクション] を選択し、[アクティブ化] または [非アクティブ化] を選択します。

> **Note**
>
> 非アクティブ化されたプローブを再度アクティブ化すると、そのプローブの請求料金が開始されます。

コマンドラインまたは API を使用してプローブをアクティブまたは非アクティブに設定する

コマンドラインまたは API を使用して、プローブをアクティブまたは非アクティブに設定できます。このコマンドは 1 つのプローブにのみ使用できます。

コマンドラインまたは API を使用してプローブをアクティブまたは非アクティブに設定するには

1. モニタの名前がわからない場合は、list-monitors を使用してモニタのリストを取得します。プローブの状態を変更するモニタの名前を書き留めておきます。
2. 前の手順で取得したモニタ名を使って get-monitor を使用します。これにより、そのモニタに関連するすべてのプローブのリストが返されます。状態を変更するプローブのプローブ ID を書き留めます。
3. update-probe を使用して、状態を変更したいプローブを `ACTIVE` または `INACTIVE` に設定します。

プローブをモニタに追加する

プローブは既存のモニタに追加できます。プローブをモニタに追加すると、新しいプローブの追加を反映して請求体系が更新されることに注意してください。

コンソールを使用してプローブをモニタに追加する

コンソールを使用してプローブをモニタに追加するには

1. https://console.aws.amazon.com/cloudwatch/ で CloudWatch コンソールを開き、[ネットワークモニタリング] の下で [Network Monitor] を選択します。
2. [ネットワークモニタ] セクションで、次のいずれかを実行します。
   - プローブの追加先のモニタの [名前] リンクを選択します。[モニタの詳細] タブを選択し、[プローブ] セクションで [プローブを追加] を選択します。
   - モニタのチェックボックスをオンにし、[アクション] を選択し、次に [プローブを追加] を選択します。
3. [プローブを追加] ページで、次の手順を実行します。
   1. [AWS ネットワークソース] で、モニタに追加するサブネットを選択します。

      > **Note**
      > 一度に追加できるプローブは 1 つだけです。モニタあたり最大 4 つのプローブを追加できます。

   2. オンプレミスネットワークの宛先 [IP アドレス] を入力します。IPv4 と IPv6 両方のアドレスがサポートされます。
   3. [詳細設定] を選択します。
   4. 宛先のネットワークの [プロトコル] を選択します。[ICMP] または [TCP] を選択できます。
   5. [プロトコル] を [TCP] にする場合は、次の情報を入力します。それ以外の場合は、次の手順に進みます。
      - ネットワークが接続に使用する [ポート] を入力します。ポートは 1 ~ 65535 の数字でなければなりません。
      - [パケットサイズ] を入力します。これは、プローブによってソースと宛先の間で送信される各パケットのサイズ (バイト単位) です。パケットサイズは 56 ~ 8500 の数値でなければなりません。
4. (オプション) このリソースの特定を容易にするために、[タグ] セクションに [キー] と [値] のペアを追加します。これにより、特定の情報で検索またはフィルタリングできるようになります。
   1. [新しいタグを追加] をクリックします。
   2. [キー] の名前と関連する [値] を入力します。

3. [新しいタグを追加] を選択して新しいタグを追加します。

   [新しいタグを追加] を選択して複数のタグを追加することも、[削除] を選択して任意のタグを削除することもできます。

5. [プローブを追加] を選択します。

   プローブのアクティブ化中は、[状態] は [保留中] になります。プローブが [アクティブ] になるまでに数分かかる場合があります。

コマンドラインまたは API を使用してプローブをモニタに追加する

コマンドラインまたは API を使用してプローブをモニタに追加できます。このコマンドを使用して、一度に追加できるプローブは 1 つだけです。

コマンドラインまたは API を使用してプローブをモニタに追加するには

1. モニタの名前がわからない場合は、list-monitors を使用してモニタのリストを取得します。プローブを追加するモニタの名前を書き留めておきます。
2. create-probe を使用してプローブをモニタに追加します。

プローブの編集

プローブがアクティブか非アクティブかに関係なく、現在使用しているプローブの情報はすべて変更することができます。プローブは、Amazon CloudWatch コンソールを使用するか、コマンドラインまたは API を使用して編集できます。

コンソールを使用したプローブの編集

コンソールを使用してプローブを編集するには

1. https://console.aws.amazon.com/cloudwatch/ で CloudWatch コンソールを開き、[ネットワークモニタリング] の下で [Network Monitor] を選択します。

   [名前] リンクを選択してモニタのダッシュボードを開きます。

2. [モニタの詳細] タブを選択します。
3. [プローブ] セクションで、編集するプローブのリンクを選択します。
4. プローブのダッシュボードページで [編集] を選択するか、[アクション] を選択してから [編集] を選択します。

5. [プローブの編集] ページで、新しい宛先プローブの [IP アドレス] を入力します。IPv4 と IPv6 両方のアドレスがサポートされます。
6. [詳細設定] を選択します。
7. ネットワークの [プロトコル] を選択します。[ICMP] または [TCP] を選択できます。
8. [プロトコル] を [TCP] にする場合は、次の情報を入力します。それ以外の場合は、次の手順に進みます。

   - ネットワークが接続に使用する [ポート] を入力します。ポートは 1 ~ 65535 の数字でなければなりません。
   - [パケットサイズ] を入力します。これは、プローブによってソースと宛先の間で送信される各パケットのサイズ (バイト単位) です。パケットサイズは 56 ~ 8500 の数値でなければなりません。
9. (オプション) このリソースの特定を容易にするために、[タグ] セクションに [キー] と [値] のペアを追加します。これにより、特定の情報で検索またはフィルタリングできるようになります。

   1. [新しいタグを追加] をクリックします。
   2. [キー] の名前と関連する [値] を入力します。
   3. [新しいタグを追加] を選択して新しいタグを追加します。

      [新しいタグを追加] を選択して複数のタグを追加することも、[削除] を選択して任意のタグを削除することもできます。
10. [Save changes] (変更の保存) をクリックします。

コマンドラインまたは API を使用したプローブの編集

コマンドラインを使用してモニタプローブを編集できます。このコマンドは 1 つのプローブにのみ使用できます。

コマンドラインまたは API を使用してプローブを編集するには

1. モニタの名前がわからない場合は、list-monitors を使用してモニタのリストを取得します。プローブの状態を変更するモニタの名前を書き留めておきます。
2. 前の手順で取得したモニタ名を使って get-monitor を使用します。これにより、そのモニタに関連するすべてのプローブのリストが返されます。編集するプローブのプローブ ID を書き留めておきます。
3. update-probe を使用してプローブの情報を変更します。

プローブの削除

将来再度必要としないことがわかっている場合は、プローブを非アクティブ化せずに削除できます。削除したプローブは復元できないため、再作成する必要があります。プローブが削除されると、そのプローブの請求は停止されます。プローブは、Amazon CloudWatch コンソール、コマンドライン、API のいずれかを使用して削除できます。

コンソールを使用したプローブの削除

コンソールを使用してプローブを削除するには

1. https://console.aws.amazon.com/cloudwatch/ で CloudWatch コンソールを開き、[ネットワークモニタリング] の下で [Network Monitor] を選択します。
2. [ネットワークモニタ] セクションで、[名前] リンクを選択してモニタダッシュボードを開きます。
3. [モニタの詳細] タブを選択します。
4. モニタのチェックボックスをオンにし、[アクション] を選択してから [削除] を選択します。
5. [プローブを削除] ダイアログボックスで、[削除] を選択し、プローブを削除することを確定します。
6. [削除] を選択してプローブの削除を確定します。

   [プローブ] セクションのプローブの [状態] に [削除中] と表示されます。削除が完了すると、そのプローブは [プローブ] セクションから削除されます。

コマンドラインまたは API を使用したプローブの削除

コマンドラインまたは API を使用してプローブを削除できます。このコマンドは 1 つのプローブにのみ使用できます。

コマンドラインまたは API を使用してプローブをアクティブまたは非アクティブに設定するには

1. モニタの名前がわからない場合は、list-monitors を使用してモニタのリストを取得します。削除するプローブが含まれているモニタの名前を書き留めておきます。
2. 前の手順で取得したモニタ名を使って get-monitor を使用します。これにより、そのモニタに関連するすべてのプローブのリストが返されます。削除するプローブのプローブ ID を書き留めておきます。
3. delete-probe を使用します。

コマンドラインまたは API を使用したリソースのタグ付けまたはタグの解除

コマンドラインまたは CLI を使用して、リソースタグを追加または更新できます。

コマンドラインまたは API を使用してネットワークモニタのタグを更新するには

- リソースのタブをリストするには、list-tags-for-resources を使用します。
- リソースにタグを付けるには、tag-resource を使用します。
- リソースのタグを解除するには、untag-resource を使用します。

Network Monitor ダッシュボード

Amazon CloudWatch Network Monitor ダッシュボードを使用すると、AWS ネットワークの状態を表示したり、ラウンドトリップタイムとパケット損失を調べたりできます。モニタと個々のプローブ両方のこれらのメトリクスを表示できます。

Network Monitor ダッシュボード

- モニタダッシュボード
- プローブダッシュボード

プローブアラーム

Amazon CloudWatch アラームは、他の CloudWatch メトリクスと同様に、Amazon CloudWatch Network Monitor メトリクスに基づいて作成できます。作成したアラームは、アラームがトリガーされたときに、ネットワークモニターダッシュボードの [モニターの詳細] セクションにあるプローブの [ステータス] 列に表示されます。ステータスは [OK] または [アラーム状態] になります。プローブのステータスが表示されない場合、そのプローブのアラームは作成されません。

例えば、Network Monitor のメトリクス `PacketLoss` に基づいてアラームを作成し、このメトリクスが選択した値を上回ったときに通知を送信するように設定できます。Network Monitor メトリクスのアラームは、他の CloudWatch メトリクスと同じガイドラインに従って設定します。

Network Monitor 用の CloudWatch アラームを作成する場合、`AWS/NetworkMonitor` の以下のメトリクスを利用できます。

- ヘルスインジケータ
- パケット損失

- RTT (ラウンドトリップタイム)

Network Monitor アラームを作成する手順については、「the section called “静的しきい値に基づいてアラームを作成する”」を参照してください。

メトリクスの時間枠の設定

両方のダッシュボードのメトリクスとイベントでは、現在の時刻から計算された 2 時間のデフォルト時間が使用されます。デフォルトを変更して、次のいずれかのプリセットを使用できます。

- 1h — 1 時間
- 2h — 2 時間
- 1d — 1 日
- 1w — 1 週間

カスタムの時間枠を設定することもできます。[カスタム] を選択し、[絶対時間] または [相対時間] を選択して、時間枠を任意の時間に設定します。相対時間は、CloudWatch のデフォルトに従って、今日の日付から 15 日前のみをサポートします。

さらに、UTC タイムゾーンまたはローカルタイムゾーンのいずれかに基づいてチャートに表示される時間を選択できます。

モニタダッシュボード

Amazon CloudWatch Network Monitor ダッシュボードを使用すると、AWS ネットワークの状態を表示したり、ラウンドトリップタイムとパケット損失を調べたりできます。Network Monitor には、モニタ用とプローブ用の両方のダッシュボードがあります。

モニタダッシュボードにアクセスするには

1. https://console.aws.amazon.com/cloudwatch/ で CloudWatch コンソールを開き、[ネットワークモニタリング] の下で [Network Monitor] を選択します。
2. [ネットワークモニタ] セクションで、[名前] リンクを選択してモニタダッシュボードを開きます。

概要

[概要] ページには、モニタの次の情報が表示されます。

- [AWS ネットワークヘルス] — [AWS ネットワークヘルス] には、AWS ネットワークのみの全体的な状態が表示されます。ステータスは [正常] または [パフォーマンス低下] のいずれかになります。[正常] ステータスは、Network Monitor が AWS ネットワークに関する問題を観察しなかったことを示します。[パフォーマンス低下] ステータスは、Network Monitor が AWS ネットワークに関する問題を観察したことを示します。このセクションのステータスバーには、デフォルト値の 1 時間にわたるネットワークの状態が表示されます。ステータスバーの任意のポイントにカーソルを合わせると、追加の詳細が表示されます。
- [プローブトラフィックの概要] — モニタのソース AWS サブネットと宛先 IP アドレス間のトラフィックの現在の状態が表示されます。[プローブトラフィックの概要] には次の情報が表示されます。
  - [アラームのプローブ] — この数字は、パフォーマンスが低下している状態にあるプローブの数を示します。アラームとして設定したメトリクスがトリガーされると、アラームがトリガーされます。Network Monitor メトリクスアラームの詳細については、「the section called “プローブアラーム”」を参照してください。
  - [パケット損失] — ソースサブネットから宛先 IP アドレスの間で失われたパケットの数。これは、送信されたパケット全体に占める割合で表されます。
  - [ラウンドトリップタイム] — ソースサブネットからのパケットが宛先 IP アドレスに到達し、ソースに戻ってくるまでにかかる時間 (ミリ秒単位)。

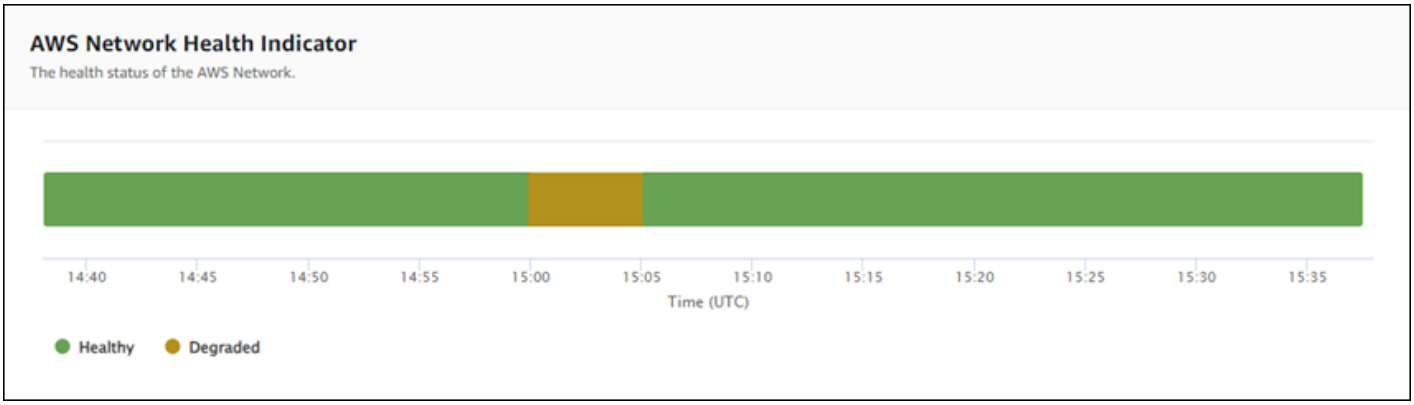
データはインタラクティブなグラフで表されるため、詳細を確認できます。

デフォルトでは、データは現在の日付と時刻から計算された 2 時間の時間枠で表示されます。ただし、時間の範囲は要件に合わせて変更することができます。詳細については、「the section called “メトリクスの時間枠の設定”」を参照してください。

メトリクスの追跡

Network Monitor ダッシュボードには、モニタとプローブがグラフィカルに表示されます。以下のグラフが利用可能です。

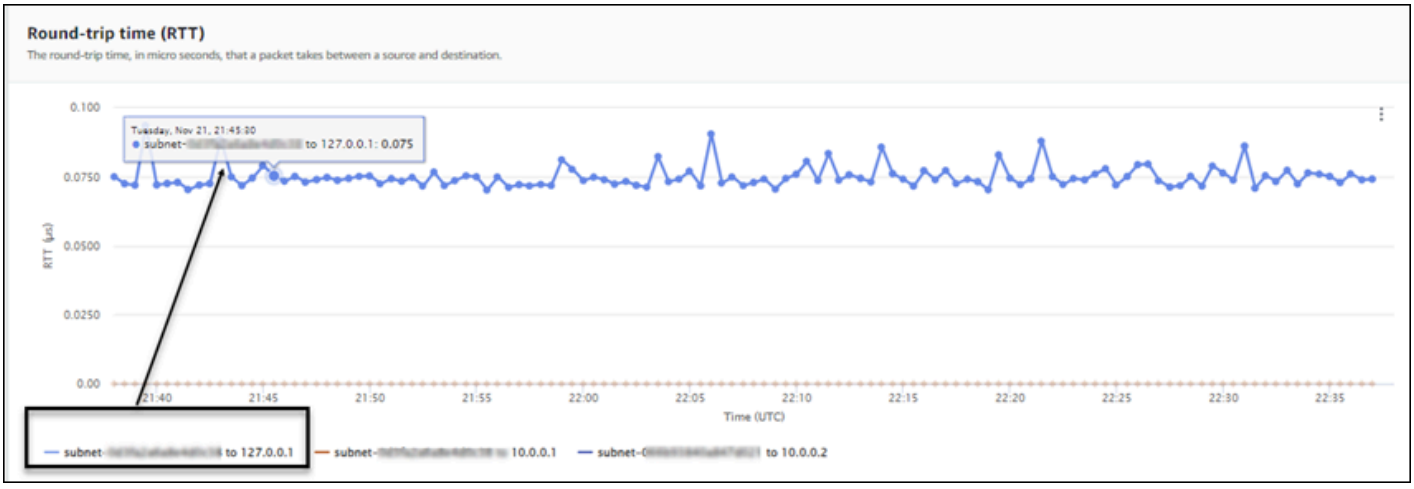
- [AWS ネットワークヘルスインジケータ] — 指定された期間における AWS ネットワークの状態を表します。ステータスは [正常] または [パフォーマンス低下] のいずれかになります。次の例では、UTC 15:00 から 15:05 UTC までの間、AWS ネットワークがパフォーマンス低下状態だったことがわかります。15:05 を過ぎると、ネットワークは正常な状態に戻りました。グラフの任意の部分にカーソルを合わせると、追加の詳細が表示されます。



> **Note**
>
> ネットワークヘルスインジケータは、プローブの状態ではなく、AWS ネットワークのみの状態を示します。

- [パケット損失] — このグラフには、モニタ内の各プローブのパケット損失の割合を示す独自の線が表示されます。ページ下部の凡例には、モニタ内の各プローブが一意になるように色分けされて表示されます。このグラフのプローブにカーソルを合わせると、ソースサブネット、宛先 IP、およびパケット損失の割合が表示されます。次の例では、サブネットから IP アドレス 127.0.0.1 へのプローブにパケット損失アラームが設定されています。このアラームは、プローブのパケット損失がしきい値を超えたときにトリガーされました。グラフにカーソルを合わせると、プローブのソースと宛先が表示され、11 月 21 日 02:41:30 にこのプローブで 30.97% のパケット損失があったことがわかります。

- [ラウンドトリップタイム] — このグラフには、各プローブの線が表示され、各プローブのラウンドトリップ時間がわかります。ページ下部の凡例には、モニタ内の各プローブが一意になるように色分けされて表示されます。このグラフのプローブにカーソルを合わせると、ソースサブネット、宛先 IP アドレス、およびラウンドトリップタイムが表示されます。次の例は、11 月 21 日火曜日の 21:45:30 に、サブネットから IP アドレス 127.0.0.1 へのプローブのラウンドトリップタイムが 0.075 秒だったことを示しています。



モニタの詳細

[モニタの詳細] ページには、プローブを含むモニタの詳細が表示されます。このページでは、タグを管理したり、プローブを追加したりできます。このページは、以下の 3 つのセクションに分かれています。

- [モニタの詳細] — このページには、モニタに関する詳細が表示されます。このセクションの情報は編集できません。ただし、[ロール名] リンクを選択すると、Network Monitor サービスリンクロールの詳細を表示できます。
- [プローブ] — このセクションには、モニタに関連するすべてのプローブのリストが表示されます。[VPC] または [サブネット ID] リンクを選択すると、Amazon VPC コンソールで VPC またはサブネットの詳細が開きます。アクティブ化または非アクティブ化するなど、プローブを変更することもできます。詳細については、「the section called “モニタとプローブの操作”」を参照してください。

  [プローブ] セクションには、プローブの [ID]、[VPC ID]、[サブネット ID]、[IP アドレス]、[プロトコル]、およびプローブの [状態] が [アクティブ] か [非アクティブ] かなど、そのモニタに設定された各プローブに関する情報が表示されます。プローブにアラームを設定している場合、そのアラームの現在の [ステータス] が表示されます。[OK: は、アラームをトリガーしたメトリクスイベントがないことを示します。[アラーム状態] は、CloudWatch で設定したメトリクスがアラームをトリガーしたことを示します。プローブのステータスが表示されていない場合、CloudWatch のアラームは設定されていません。作成できる Network Monitor プローブアラームのタイプについては、「the section called “プローブアラーム”」を参照してください。
- [タグ] — モニタの現在のタグが表示されます。タグを追加または削除するには、[タグを管理] を選択します。これにより [プローブの編集] ページが開きます。タグの編集の詳細については、「the section called “モニタの編集”」を参照してください。

プローブダッシュボード

Amazon CloudWatch Network Monitor ダッシュボードを使用すると、AWS ネットワークの状態や、特定のプローブの具体的なラウンドトリップタイムとパケット損失に関する情報を表示できます。[概要] と [プローブの詳細] の 2 つのプローブダッシュボードがあります。

CloudWatch アラームを作成して、パケット損失とラウンドトリップタイムのメトリクスしきい値を設定することができます。メトリクスしきい値に達すると、CloudWatch アラームから通知が送信されます。プローブ作成の詳細については、「the section called “プローブアラーム” 」を参照してください。

プローブダッシュボードにアクセスするには

1. https://console.aws.amazon.com/cloudwatch/ で CloudWatch コンソールを開き、[ネットワークモニタリング] の下で [Network Monitor] を選択します。
2. [ネットワークモニタ] セクションで、[名前] リンクを選択してモニタダッシュボードを開きます。
3. [ID] リンクを選択すると、そのプローブのダッシュボードが表示されます。

概要

[概要] ページには、プローブに関する次の情報が表示されます。

- [AWS ネットワークヘルスインジケータの詳細] — AWS ネットワークのみの全体的な状態が表示されます。ステータスは [正常] または [パフォーマンス低下] のいずれかになります。[パフォーマンス低下] ステータスは、AWS ネットワークに問題があることを示します。プローブに問題があるかどうかを示すものではありません。
- [パケット損失] — ソースサブネットから宛先 IP アドレスの間で失われたこのプローブのパケットの数。
- [ラウンドトリップタイム] — ソースサブネットからのパケットが宛先 IP アドレスに到達し、ソースに戻ってくるまでにかかる時間 (ミリ秒単位)。

プローブの詳細

[プローブの詳細] ページには、プローブに関する詳細が表示されます。このページではプローブを編集できます。詳細については、「the section called “モニタとプローブの操作”」を参照してください。

- [プローブの詳細] — このページには、プローブに関する一般的な情報が表示されます。このセクションの情報は編集できません。
- [プローブの送信元と送信先] — このセクションにはプローブに関する詳細が表示されます。[VPC] または [サブネット ID] リンクを選択すると、Amazon VPC コンソールで VPC またはサブネットの詳細が開きます。アクティブ化または非アクティブ化するなど、プローブを変更することもできます。
- [タグ] — モニタの現在のタグが表示されます。タグを追加または削除するには、[タグを管理] を選択します。これにより [プローブの編集] ページが開きます。タグの編集の詳細については、「the section called “プローブの編集 ”」を参照してください。

Network Monitor のクォータ

Network Monitor のクォータは次のとおりです。

クォータ	デフォルト値	引き上げ可能
1 つの AWS リージョンのアカウントあたりのモニタの最大数	100	はい
モニタあたりのプローブの最大数	24	はい
モニタ 1 つのサブネットあたりの最大プローブ数	4	はい

Network Monitor のデータセキュリティとデータ保護

AWS でのクラウドセキュリティは最優先事項です。AWS のユーザーは、セキュリティを最も重視する組織の要件を満たすように構築されたデータセンターとネットワークアーキテクチャを利用できます。

セキュリティは、AWS とユーザーの間の責任共有です。責任共有モデル では、これをクラウドのセキュリティおよびクラウド内のセキュリティとして説明しています。

- クラウドのセキュリティ - AWS は、AWS クラウド で AWS のサービスを実行するインフラストラクチャを保護する責任を担います。また、AWS は、ユーザーが安全に使用できるサービスも提供します。AWS コンプライアンスプログラム の一環として、サードパーティーの監査が定期的にセキュリティの有効性をテストおよび検証しています。Amazon CloudWatch Network Monitor に適用するコンプライアンスプログラムの詳細については、「コンプライアンスプログラムによる対象範囲の AWS のサービス」「」を参照してください。
- クラウド内のセキュリティ - ユーザーの責任は、使用する AWS サービスに応じて異なります。また、お客様は、データの機密性、会社の要件、適用される法律や規制など、その他の要因についても責任を負います。

このドキュメントは、CloudWatch Network Monitor を使用する際の責任共有モデルの適用方法を理解するのに役立ちます。以下のトピックでは、セキュリティおよびコンプライアンスの目的を達

成に向けて CloudWatch Network Monitor を設定する方法について説明します。また、CloudWatch Network Monitor リソースのモニタリングや保護に他の AWS のサービスを利用する方法についても説明します。

トピック

- Amazon CloudWatch Network Monitor におけるデータ保護
- Amazon CloudWatch Network Monitor のインフラストラクチャセキュリティ

Amazon CloudWatch Network Monitor におけるデータ保護

AWS 責任共有モデルは、Amazon CloudWatch Network Monitor のデータ保護に適用されます。このモデルで説明されているように、AWS は、AWS クラウド のすべてを実行するグローバルインフラストラクチャを保護するがあります。お客様は、このインフラストラクチャでホストされているコンテンツに対する管理を維持する責任があります。また、使用する AWS のサービスのセキュリティ設定と管理タスクもユーザーの責任となります。データプライバシーの詳細については、「データプライバシーのよくある質問」を参照してください。欧州でのデータ保護の詳細については、AWS セキュリティブログに投稿された記事「AWS 責任共有モデルおよび GDPR」を参照してください。

データを保護するため、AWS アカウント 認証情報を保護し、AWS IAM Identity Center または AWS Identity and Access Management (IAM) を使用して個々のユーザーをセットアップすることをお勧めします。この方法により、それぞれのジョブを遂行するために必要な権限のみが各ユーザーに付与されます。また、次の方法でデータを保護することもお勧めします:

- 各アカウントで多要素認証 (MFA) を使用します。
- SSL/TLS を使用して AWS リソースと通信します。TLS 1.2 は必須であり TLS 1.3 がお勧めです。
- AWS CloudTrail で API とユーザーアクティビティロギングをセットアップします。
- AWS のサービス 内のすべてのデフォルトセキュリティ管理に加え、AWS 暗号化ソリューションを使用します。
- Amazon Macie などの高度なマネージドセキュリティサービスを使用します。これらは、Amazon S3 に保存されている機密データの検出と保護を支援します。
- コマンドラインインターフェイスまたは API により AWS にアクセスするときに FIPS 140-2 検証済み暗号化モジュールが必要な場合は、FIPS エンドポイントを使用します。利用可能な FIPS エンドポイントの詳細については、「連邦情報処理規格 (FIPS) 140-2」を参照してください。

お客様の E メールアドレスなどの極秘または機密情報は、タグ、または名前フィールドなどの自由形式のテキストフィールドに配置しないことを強くお勧めします。これは、コンソール、API、AWS

CLI、または AWS SDK によって CloudWatch Network Monitor や他の AWS のサービス サービスを使用する場合も同様です。名前に使用する自由記述のテキストフィールドやタグに入力したデータは、課金や診断ログに使用される場合があります。外部サーバーへの URL を提供する場合は、そのサーバーへのリクエストを検証するための認証情報を URL に含めないように強くお勧めします。

Amazon CloudWatch Network Monitor のインフラストラクチャセキュリティ

マネージドサービスである Amazon CloudWatch Network Monitor は、「Amazon Web Services: セキュリティプロセスの概要」ホワイトペーパーに記載されている AWS グローバルネットワークセキュリティの手順で保護されています。

AWS が発行した API 呼び出しを使用して、ネットワーク経由で CloudWatch Network Monitor にアクセスします。クライアントで Transport Layer Security (TLS) 1.0 以降がサポートされている必要があります。TLS 1.2 以降が推奨されています。また、DHE (Ephemeral Diffie-Hellman) や ECDHE (Elliptic Curve Ephemeral Diffie-Hellman) などの Perfect Forward Secrecy (PFS) を使用した暗号スイートもクライアントでサポートされている必要があります。これらのモードは、Java 7 以降など、ほとんどの最新システムでサポートされています。

また、リクエストには、アクセスキー ID と、IAM プリンシパルに関連付けられているシークレットアクセスキーを使用して署名する必要があります。または、AWS Security Token Service (AWS STS) を使用して、一時的なセキュリティ認証情報を生成し、リクエストに署名することもできます。

Amazon CloudWatch Network Monitor 用 Identity and Access Management

AWS Identity and Access Management (IAM) は、AWS リソースへのアクセスを管理者が安全に制御するために役立つ AWS のサービスです。IAM の管理者は、誰を認証 (サインインを許可) し、誰に CloudWatch Network Monitor リソースの使用を承認する (アクセス許可を持たせる) かを制御します。IAM は、AWSのサービスで追加料金は発生しません。IAM の機能を使用して、他のユーザー、サービス、およびアプリケーションが完全にまたは制限付きでお客様のAWSリソースを使用できるようにします。その際、お客様のセキュリティ認証情報は共有されません。

デフォルトでは、IAM ユーザーには、AWS リソースを作成、表示、変更するためのアクセス許可はありません。IAM ユーザーがグローバルネットワークなどのリソースにアクセスし、タスクを実行できるようにするには、次の操作を行う必要があります。

- 必要な特定のリソースと API アクションを使用するアクセス許可をユーザーに付与する IAM ポリシーを作成します。
- IAM ユーザーまたは IAM ユーザーが属するグループに、そのポリシーをアタッチします。

ポリシーをユーザーまたはユーザーのグループにアタッチする場合、ポリシーによって特定リソースの特定タスクを実行するユーザーの権限が許可または拒否されます。

条件キー

Condition 要素 (または条件ブロック) を使用すると、ステートメントが有効な条件を指定できます。Condition 要素はオプションです。イコールや以下などの条件演算子を使用する条件表現を構築して、リクエスト内に値のあるポリシーの条件に一致させることができます。詳細については、「AWS Identity and Access Management ユーザーガイド」の「IAM JSON ポリシー要素: 条件演算子」を参照してください。

1 つのステートメントに複数の Condition 要素を指定するか、1 つの Condition 要素に複数のキーを指定すると、AWS は AND 論理演算子を使用してそれらを評価します。単一の条件キーに複数の値を指定すると、AWS は OR 論理演算子を使用して条件を評価します。ステートメントの権限が付与される前にすべての条件が満たされる必要があります。

条件を指定する際にプレースホルダー変数も使用できます。例えば IAM ユーザーに、IAM ユーザー名がタグ付けされている場合のみリソースにアクセスできる権限を付与することができます。

タグを CloudWatch Network Monitor リソースにアタッチすることも、Cloud WAN へのリクエストでタグを渡すこともできます。タグに基づいてアクセスを制御するには、aws:ResourceTag/key-name、aws:RequestTag/key-name、または aws:TagKeys の条件キーを使用して、ポリシーの条件要素でタグ情報を提供します。詳細については、「AWS Identity and Access Management ユーザーガイド」の「IAM JSON ポリシー要素: 条件演算子」を参照してください。

すべての AWS グローバル条件キーを確認するには、「AWS Identity and Access Management ユーザーガイド」の「AWS グローバル条件コンテキストキー」を参照してください。

コアネットワークリソースのタグ付け

タグは、ユーザーまたは AWS が AWS リソースに割り当てるメタデータラベルです。各タグは、キーと値から構成されます。ユーザーが割り当てるタグは、ユーザーがキーと値を定義します。たとえば、1 つのリソースのキーを purpose と定義し、値を test と定義します。タグは、以下のことに役立ちます。

- AWS リソースの特定と整理。多くの AWS のサービスではタグ付けがサポートされるため、さまざまなサービスからリソースに同じタグを割り当てて、リソースの関連を示すことができます。
- AWS リソースへのアクセスを制御します。詳細については、「AWS Identity and Access Management ユーザーガイド」の「タグを使用した AWS リソースへのアクセスの制御」を参照してください。

IAM と Amazon CloudWatch Network Monitor の連携

IAM を使用して CloudWatch Network Monitor へのアクセスを管理する前に、CloudWatch Network Monitor で利用できる IAM の機能を確認してください。

Amazon CloudWatch Network Monitor で利用できる IAM 機能

IAM の機能	CloudWatch Network Monitor のサポート
アイデンティティベースのポリシー	Yes
リソースベースのポリシー	No
ポリシーアクション	Yes
ポリシーリソース	Yes
ポリシー条件キー	Yes
ACL	No
ABAC (ポリシー内のタグ)	部分的
一時的な認証情報	Yes
プリンシパル権限	Yes
サービスロール	いいえ
サービスリンクロール	はい

大部分の IAM 機能が CloudWatch Network Monitor やその他の AWS サービスとどのように連携するかの概要については、「IAM ユーザーガイド」の「IAM と連携する AWS のサービス」を参照してください。

Amazon CloudWatch Network Monitor のアイデンティティベースポリシー

アイデンティティベースポリシーをサポートする	Yes

アイデンティティベースポリシーは、IAM ユーザー、ユーザーグループ、ロールなど、アイデンティティにアタッチできる JSON 権限ポリシードキュメントです。これらのポリシーは、ユーザーとロールが実行できるアクション、リソース、および条件をコントロールします。アイデンティティベースのポリシーを作成する方法については、『IAM ユーザーガイド』の「IAM ポリシーの作成」を参照してください。

IAM アイデンティティベースのポリシーでは、許可または拒否するアクションとリソース、およびアクションを許可または拒否する条件を指定できます。プリンシパルは、それが添付されているユーザーまたはロールに適用されるため、アイデンティティベースのポリシーでは指定できません。JSON ポリシーで使用できるすべての要素については、「IAM ユーザーガイド」の「IAM JSON ポリシーの要素のリファレンス」を参照してください。

CloudWatch Network Monitor のアイデンティティベースのポリシー例

CloudWatch Network Monitor のアイデンティティベースのポリシー例については、「Amazon CloudWatch のアイデンティティベースのポリシー例」を参照してください。

CloudWatch Network Monitor 内のリソースベースのポリシー

リソースベースのポリシーのサポート	No

リソースベースのポリシーは、リソースに添付する JSON ポリシードキュメントです。リソースベースのポリシーには例として、IAM ロールの信頼ポリシーや Amazon S3 バケットポリシーがあげられます。リソースベースのポリシーをサポートするサービスでは、サービス管理者はポリシーを使用して特定のリソースへのアクセスを制御できます。ポリシーがアタッチされているリソースの場合、指定されたプリンシパルがそのリソースに対して実行できるアクションと条件は、ポリシーによって定義されます。リソースベースのポリシーでは、プリンシパルを指定する必要があります。プリンシパルには、アカウント、ユーザー、ロール、フェデレーションユーザー、または AWS のサービス を含めることができます。

クロスアカウントアクセスを有効にするには、全体のアカウント、または別のアカウントの IAM エンティティを、リソースベースのポリシーのプリンシパルとして指定します。リソースベースのポリシーにクロスアカウントのプリンシパルを追加しても、信頼関係は半分しか確立されない点に注意してください。プリンシパルとリソースが異なる AWS アカウント にある場合、信頼できるアカウントの IAM 管理者は、リソースにアクセスするための権限をプリンシパルエンティティ (ユーザーまたはロール) に付与する必要もあります。IAM 管理者は、アイデンティティベースのポリシーをエンティティにアタッチすることで権限を付与します。ただし、リソースベースのポリシーで、同じアカ

ウントのプリンシパルへのアクセス権が付与されている場合は、アイデンティティベースのポリシーを追加する必要はありません。詳細については、『IAM ユーザーガイド』の「IAM ロールとリソースベースのポリシーとの相違点」を参照してください。

CloudWatch Network Monitor のポリシーアクション

ポリシーアクションに対するサポート	Yes

管理者は AWS JSON ポリシーを使用して、誰が何にアクセスできるかを指定できます。つまり、どのプリンシパルがどんなリソースにどんな条件でアクションを実行できるかということです。

JSON ポリシーのAction要素には、ポリシー内のアクセスを許可または拒否するために使用できるアクションが記述されます。ポリシーアクションの名前は通常、関連する AWSAPI オペレーションと同じです。一致する API オペレーションのない権限のみのアクションなど、いくつかの例外があります。また、ポリシーに複数アクションが必要なオペレーションもあります。これらの追加アクションは、 依存アクション と呼ばれます。

このアクションは、関連付けられたオペレーションを実行するための権限を付与するポリシーで使用されます。

CloudWatch Network Monitor アクションのリストを確認するには、「サービス認証リファレンス」の「Amazon CloudWatch Network Monitor で定義されるアクション」を参照してください。

CloudWatch Network Monitor のポリシーアクションは、アクションの前に次のプレフィックスを使用します。

```
networkmonitor
```

単一のステートメントで複数のアクションを指定するには、アクションをカンマで区切ります。

```
"Action": [
      "networkmonitor:action1",
      "networkmonitor:action2"
         ]
```

CloudWatch Network Monitor のアイデンティティベースのポリシー例については、「Amazon CloudWatch のアイデンティティベースのポリシー例」を参照してください。

CloudWatch Network Monitor のポリシーリソース

ポリシーリソースに対するサポート	Yes

管理者は AWS JSON ポリシーを使用して、誰が何にアクセスできるかを指定できます。つまり、どのプリンシパルがどんなリソースにどんな条件でアクションを実行できるかということです。

Resource JSON ポリシー要素は、アクションが適用されるオブジェクトを指定します。ステートメントには、Resource または NotResource要素を含める必要があります。ベストプラクティスとして、Amazon リソースネーム (ARN) を使用してリソースを指定します。これは、リソースレベルの権限と呼ばれる特定のリソースタイプをサポートするアクションに対して実行できます。

オペレーションのリスト化など、リソースレベルの権限をサポートしないアクションの場合は、ステートメントがすべてのリソースに適用されることを示すために、ワイルドカード (*) を使用します。

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

CloudWatch Network Monitor リソースのタイプとその ARN のリストを確認するには、「サービス認証リファレンス」の「Amazon CloudWatch Network Monitor で定義されるリソースタイプ」を参照してください。どのアクションで各リソースの ARN を指定できるかについては、「Amazon CloudWatch Network Monitor で定義されるアクション」を参照してください。

Amazon CloudWatch Network Monitor のポリシー条件キー

サービス固有のポリシー条件キーのサポート	はい

管理者は AWS JSON ポリシーを使用して、誰が何にアクセスできるかを指定できます。つまり、どのプリンシパルがどんなリソースにどんな条件でアクションを実行できるかということです。

Condition 要素 (または Condition ブロック) を使用すると、ステートメントが有効な条件を指定できます。Condition 要素はオプションです。イコールや未満などの 条件演算子 を使用して条件式を作成することで、ポリシーの条件とリクエスト内の値を一致させることができます。

1 つのステートメントに複数の Condition 要素を指定するか、1 つの Condition 要素に複数のキーを指定すると、AWS は AND 論理演算子を使用してそれらを評価します。単一の条件キーに複数

の値を指定すると、AWS は 0R 論理演算子を使用して条件を評価します。ステートメントの権限が付与される前にすべての条件が満たされる必要があります。

条件を指定する際にプレースホルダー変数も使用できます。例えば IAM ユーザーに、IAM ユーザー名がタグ付けされている場合のみリソースにアクセスできる権限を付与することができます。詳細については、『IAM ユーザーガイド』の「IAM ポリシーの要素: 変数およびタグ」を参照してください。

AWS はグローバル条件キーとサービス固有の条件キーをサポートしています。すべての AWS グローバル条件キーを確認するには、IAM ユーザーガイド の「AWS グローバル条件コンテキストキー」を参照してください。

CloudWatch Network Monitor の条件キーのリストを確認するには、「サービス認証リファレンス」の「Amazon CloudWatch Network Monitor の条件キー」を参照してください。どのアクションやリソースで条件キーを使用できるかについては、「Amazon CloudWatch Network Monitor で定義されるアクション」を参照してください。

CloudWatch Network Monitor の ACL

ACL のサポート	No

アクセスコントロールリスト (ACL) は、どのプリンシパル (アカウントメンバー、ユーザー、またはロール) がリソースにアクセスするための権限を持つかを制御します。ACL はリソースベースのポリシーに似ていますが、JSON ポリシードキュメント形式は使用しません。

CloudWatch Network Monitor での ABAC

ABAC (ポリシー内のタグ) のサポート	部分的

属性ベースのアクセスコントロール (ABAC) は、属性に基づいて権限を定義する認可戦略です。AWS では、属性は タグ と呼ばれます。タグは、IAM エンティティ (ユーザーまたはロール)、および多数の AWS リソースにアタッチできます。エンティティとリソースのタグ付けは、ABAC の最初の手順です。その後、プリンシパルのタグがアクセスしようとしているリソースのタグと一致した場合に操作を許可するように ABAC ポリシーを設計します。

ABAC は、急成長する環境やポリシー管理が煩雑になる状況で役立ちます。

タグに基づいてアクセスを管理するには、aws:ResourceTag/*key-name*、aws:RequestTag/*key-name*、または aws:TagKeys の条件キーを使用して、ポリシーの条件要素でタグ情報を提供します。

サービスがすべてのリソースタイプに対して 3 つの条件キーすべてをサポートする場合、そのサービスの値ははいです。サービスが一部のリソースタイプに対してのみ 3 つの条件キーのすべてをサポートする場合、値は「部分的」になります。

ABAC の詳細については、『IAM ユーザーガイド』の「ABAC とは?」を参照してください。ABAC をセットアップするステップを説明するチュートリアルについては、「IAM ユーザーガイド」の「属性に基づくアクセスコントロール (ABAC) を使用する」を参照してください。

CloudWatch Network Monitor での一時的な認証情報の使用

一時的な認証情報のサポート	Yes

AWS のサービスには、一時認証情報を使用してサインインしても機能しないものがあります。一時的な認証情報を利用できる AWS のサービス を含めた詳細情報については、『IAM ユーザーガイド』の「IAM と連携する AWS のサービス」を参照してください。

ユーザー名とパスワード以外の方法で AWS Management Consoleにサインインする場合は、一時認証情報を使用していることになります。例えば、会社のシングルサインオン (SSO) リンクを使用して AWS にアクセスすると、そのプロセスは自動的に一時認証情報を作成します。また、ユーザーとしてコンソールにサインインしてからロールを切り替える場合も、一時的な認証情報が自動的に作成されます。ロールの切り替えに関する詳細については、「IAM ユーザーガイド」の「ロールへの切り替え (コンソール)」を参照してください。

一時認証情報は、AWS CLI または AWSAPI を使用して手動で作成できます。作成後、一時認証情報を使用して AWSにアクセスできるようになります。AWS は、長期的なアクセスキーを使用する代わりに、一時認証情報を動的に生成することをお勧めします。詳細については、「IAM の一時的セキュリティ認証情報」を参照してください。

CloudWatch Network Monitor のクロスサービスプリンシパル許可

フォワードアクセスセッション (FAS) をサポート	Yes

IAM ユーザーまたはロールを使用して AWS でアクションを実行するユーザーは、プリンシパルとみなされます。一部のサービスを使用する際に、アクションを実行してから、別のサービスの別のアクションを開始することがあります。FAS は、AWS のサービスを呼び出すプリンシパルの権限を、AWS のサービスのリクエストと合わせて使用し、ダウンストリームのサービスに対してリクエストを行います。FAS リクエストは、サービスが、完了するために他の AWS のサービス またはリソースとのやりとりを必要とするリクエストを受け取ったときにのみ行われます。この場合、両方のアクションを実行するためのアクセス許可が必要です。FAS リクエストを行う際のポリシーの詳細については、「転送アクセスセッション」を参照してください。

CloudWatch Network Monitor のサービスロール

サービスロールのサポート	いいえ

サービスロールとは、サービスがユーザーに代わってアクションを実行するために引き受ける IAM ロールです。IAM 管理者は、IAM 内からサービスロールを作成、変更、削除できます。詳細については、『IAM ユーザーガイド』の「AWS のサービス に権限を委任するロールの作成」を参照してください。

> **Warning**
>
> サービスロールのアクセス許可を変更すると、CloudWatch Network Monitor の機能が破損する可能性があります。CloudWatch Network Monitor が指示する場合以外は、サービスロールを編集しないでください。

CloudWatch Network Monitor のサービスリンクロールの使用

サービスリンクロールのサポート	Yes

サービスリンクロールは、AWS のサービス にリンクされているサービスロールの一種です。サービスがロールを引き受け、ユーザーに代わってアクションを実行できるようになります。サービスリンクロールは、AWS アカウント に表示され、サービスによって所有されます。IAM 管理者は、サービスにリンクされたロールの権限を表示できますが、編集することはできません。

サービスにリンクされたロールの作成または管理の詳細については、「IAM と提携する AWS のサービス」を参照してください。表の中から、[Service-linked role] (サービスにリンクされたロール) 列に

Yes と記載されたサービスを見つけます。サービスにリンクされたロールに関するドキュメントをサービスで表示するには、[Yes] リンクを選択します。

CloudWatch Network Monitor のアイデンティティベースのポリシー例

デフォルトでは、ユーザーおよびロールには、CloudWatch Network Monitor リソースを作成または変更するアクセス許可がありません。また、AWS Management Console、AWS Command Line Interface (AWS CLI)、または AWS API を使用してタスクを実行することもできません。IAM 管理者は、リソースで必要なアクションを実行するための権限をユーザーに付与する IAM ポリシーを作成できます。その後、管理者はロールに IAM ポリシーを追加し、ユーザーはロールを引き受けることができます。

これらサンプルの JSON ポリシードキュメントを使用して、IAM アイデンティティベースのポリシーを作成する方法については、『IAM ユーザーガイド』の「IAM ポリシーの作成」を参照してください。

CloudWatch が定義するアクションとリソースタイプ (リソースタイプごとの ARN の形式を含む) の詳細については、「サービス認証リファレンス」の「Amazon CloudWatch Network Monitor のアクション、リソース、および条件キー」を参照してください。

トピック

- ポリシーのベストプラクティス
- CloudWatch Network Monitor コンソールの使用
- 自分の権限の表示をユーザーに許可する
- CloudWatch Network Monitor のアイデンティティとアクセスのトラブルシューティング

ポリシーのベストプラクティス

アイデンティティベースのポリシーは、アカウント内で、CloudWatch Network Monitor リソースを作成、アクセス、または削除できるかどうかを決定します。これらのアクションを実行すると、AWS アカウント に料金が発生する可能性があります。アイデンティティベースポリシーを作成したり編集したりする際には、以下のガイドラインと推奨事項に従ってください:

- AWS マネージドポリシーを使用して開始し、最小特権の権限に移行する – ユーザーとワークロードへの権限の付与を開始するには、多くの一般的なユースケースのために権限を付与する AWS マネージドポリシーを使用します。これらは AWS アカウントで使用できます。ユースケースに応じた AWS カスタマーマネージドポリシーを定義することで、権限をさらに減らすことをお勧めし

ます。詳細については、『IAM ユーザーガイド』の「AWS マネージドポリシー」または「AWS ジョブ機能の管理ポリシー」を参照してください。

- 最小特権を適用する – IAM ポリシーで権限を設定するときは、タスクの実行に必要な権限のみを付与します。これを行うには、特定の条件下で特定のリソースに対して実行できるアクションを定義します。これは、最小特権権限とも呼ばれています。IAM を使用して権限を適用する方法の詳細については、『IAM ユーザーガイド』の「IAM でのポリシーと権限」を参照してください。
- IAM ポリシーで条件を使用してアクセスをさらに制限する - ポリシーに条件を追加して、アクションやリソースへのアクセスを制限できます。例えば、ポリシー条件を記述して、すべてのリクエストを SSL を使用して送信するように指定できます。また、AWS CloudFormation などの特定の AWS のサービスを介して使用する場合、条件を使ってサービスアクションへのアクセス権を付与することもできます。詳細については、『IAM ユーザーガイド』の [IAM JSON policy elements: Condition] (IAM JSON ポリシー要素：条件) を参照してください。
- IAM Access Analyzer を使用して IAM ポリシーを検証し、安全で機能的な権限を確保する - IAM Access Analyzer は、新規および既存のポリシーを検証して、ポリシーが IAM ポリシー言語 (JSON) および IAM のベストプラクティスに準拠するようにします。IAM アクセスアナライザーは 100 を超えるポリシーチェックと実用的な推奨事項を提供し、安全で機能的なポリシーの作成をサポートします。詳細については、『IAM ユーザーガイド』の「IAM Access Analyzer ポリシーの検証」を参照してください。
- 多要素認証 (MFA) を要求する – AWS アカウント で IAM ユーザーまたはルートユーザーを要求するシナリオがある場合は、セキュリティを強化するために MFA をオンにします。API オペレーションが呼び出されるときに MFA を必須にするには、ポリシーに MFA 条件を追加します。詳細については、『IAM ユーザーガイド』の「MFA 保護 API アクセスの設定」を参照してください。

IAM でのベストプラクティスの詳細については、『IAM ユーザーガイド』の「IAM でのセキュリティのベストプラクティス」を参照してください。

CloudWatch Network Monitor コンソールの使用

Amazon CloudWatch Network Monitor コンソールにアクセスするには、最小限のアクセス許可が必要です。これらのアクセス許可により、AWS アカウント の CloudWatch Network Monitor リソースのリストと詳細を表示できます。最小限必要なアクセス許可よりも制限が厳しいアイデンティティベースのポリシーを作成すると、そのポリシーを持つエンティティ (ユーザーまたはロール) ではコンソールが意図したとおりに機能しません。

AWS CLI または AWS API のみを呼び出すユーザーには、最小限のコンソール権限を付与する必要はありません。代わりに、実行しようとしている API オペレーションに一致するアクションのみへのアクセスを許可します。

ユーザーとロールが引き続き CloudWatch Network Monitor コンソールを使用できるようにするには、エンティティに CloudWatch Network Monitor *ConsoleAccess* または *ReadOnly* AWS マネージドポリシーもアタッチします。詳細については、「IAM ユーザーガイド」の「ユーザーへのアクセス許可の追加」を参照してください。

自分の権限の表示をユーザーに許可する

この例では、ユーザーアイデンティティにアタッチされたインラインおよびマネージドポリシーの表示を IAM ユーザーに許可するポリシーの作成方法を示します。このポリシーには、コンソールで、または AWS CLI か AWS API を使用してプログラム的に、このアクションを完了する権限が含まれています。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ViewOwnUserInfo",
            "Effect": "Allow",
            "Action": [
                "iam:GetUserPolicy",
                "iam:ListGroupsForUser",
                "iam:ListAttachedUserPolicies",
                "iam:ListUserPolicies",
                "iam:GetUser"
            ],
            "Resource": ["arn:aws:iam::*:user/${aws:username}"]
        },
        {
            "Sid": "NavigateInConsole",
            "Effect": "Allow",
            "Action": [
                "iam:GetGroupPolicy",
                "iam:GetPolicyVersion",
                "iam:GetPolicy",
                "iam:ListAttachedGroupPolicies",
                "iam:ListGroupPolicies",
                "iam:ListPolicyVersions",
                "iam:ListPolicies",
                "iam:ListUsers"
            ],
            "Resource": "*"
        }
    ]
```

```
}
```

CloudWatch Network Monitor のアイデンティティとアクセスのトラブルシューティング

以下の情報は、CloudWatch Network Monitor と IAM の使用時に発生する可能性がある一般的な問題の診断や修復に役立ちます。

トピック

- CloudWatch Network Monitor でアクションを実行する権限がない
- iam: PassRole を実行する権限がありません
- 自分の AWS アカウント以外のユーザーに CloudWatch Network Monitor リソースへのアクセスを許可したい

CloudWatch Network Monitor でアクションを実行する権限がない

「I am not authorized to perform an action in Amazon Bedrock」というエラーが表示された場合、そのアクションを実行できるようにポリシーを更新する必要があります。

次のエラー例は、mateojackson IAM ユーザーがコンソールを使用して、ある *my-example-widget* リソースに関する詳細情報を表示しようとしたことを想定して、その際に必要なnetworkmonitor:*GetWidget* アクセス許可を持っていない場合に発生するものです。

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 networkmonitor:GetWidget on resource: my-example-widget
```

この場合、networkmonitor:*GetWidget* アクションを使用して *my-example-widget*リソースへのアクセスを許可するように、mateojackson ユーザーのポリシーを更新する必要があります。

サポートが必要な場合は、AWS 管理者に問い合わせてください。管理者とは、サインイン認証情報を提供した担当者です。

iam: PassRole を実行する権限がありません

iam:PassRole アクションを実行する権限がないというエラーが表示された場合は、ポリシーを更新して CloudWatch Network Monitor にロールを渡すことができるようにする必要があります。

一部の AWS のサービスでは、新しいサービスロールまたはサービスリンクロールを作成せずに、既存のロールをサービスに渡すことが許可されています。そのためには、サービスにロールを渡す権限が必要です。

以下の例に示すエラーは、marymajor という名前の IAM ユーザーがコンソールを使用して CloudWatch Network Monitor でアクションを実行しようとした場合に発生します。ただし、このアクションをサービスが実行するには、サービスロールから付与された権限が必要です。Mary には、ロールをサービスに渡す権限がありません。

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

この場合、Mary のポリシーを更新してメアリーに iam:PassRole アクションの実行を許可する必要があります。

サポートが必要な場合は、AWS 管理者に問い合わせてください。管理者とは、サインイン認証情報を提供した担当者です。

自分の AWS アカウント以外のユーザーに CloudWatch Network Monitor リソースへのアクセスを許可したい

他のアカウントのユーザーや組織外の人が、リソースにアクセスするために使用できるロールを作成できます。ロールの引き受けを委託するユーザーを指定できます。リソースベースのポリシーまたはアクセス制御リスト (ACL) をサポートするサービスの場合、それらのポリシーを使用して、リソースへのアクセスを付与できます。

詳細については、以下を参照してください。

- CloudWatch Network Monitor でこれらの機能がサポートされているかどうかを確認するには、「Amazon CloudWatch と IAM の連携方法」を参照してください。
- 所有している AWS アカウント全体のリソースへのアクセス権を提供する方法については、「IAM ユーザーガイド」の「所有している別の AWS アカウントへのアクセス権を IAM ユーザーに提供」を参照してください。
- サードパーティーの AWS アカウント にリソースへのアクセス権を提供する方法については、『IAM ユーザーガイド』の「第三者が所有する AWS アカウント へのアクセス権を付与する」を参照してください。
- ID フェデレーションを介してアクセスを提供する方法については、『IAM ユーザーガイド』の「外部で認証されたユーザー (ID フェデレーション) へのアクセス権限」を参照してください。
- クロスアカウントアクセスでのロールとリソースベースのポリシーの使用の違いの詳細については、『IAM ユーザーガイド』の「IAM ロールとリソースベースのポリシーとの相違点」を参照してください。

CloudWatch Network Monitor の AWS マネージドポリシー

ユーザー、グループ、ロールにアクセス許可を追加するには、自分でポリシーを作成するよりも、AWS 管理ポリシーを使用する方が簡単です。チームに必要な権限のみを提供する IAM カスタマーマネージドポリシーを作成するには、時間と専門知識が必要です。すぐに使用を開始するために、AWS マネージドポリシーを使用できます。これらのポリシーは、一般的なユースケースを対象範囲に含めており、AWS アカウントで利用できます。AWS マネージドポリシーの詳細については、「IAM ユーザーガイド」の「AWS マネージドポリシー」を参照してください。

AWS のサービスは、AWS マネージドポリシーを維持および更新します。AWS マネージドポリシーの許可を変更することはできません。サービスでは、新しい機能を利用できるようにするために、AWS マネージドポリシーに権限が追加されることがあります。この種類の更新は、ポリシーがアタッチされている、すべてのアイデンティティ (ユーザー、グループおよびロール) に影響を与えます。新しい機能が立ち上げられた場合や、新しいオペレーションが使用可能になった場合に、各サービスが AWS マネージドポリシーを更新する可能性が最も高くなります。サービスは、AWS マネージドポリシーから権限を削除しないため、ポリシーの更新によって既存の権限が破棄されることはありません。

さらに、AWS では、複数のサービスにまたがるジョブ機能のためのマネージドポリシーもサポートしています。例えば、`ReadOnlyAccess` AWS マネージドポリシーでは、すべての AWS のサービスおよびリソースへの読み取り専用アクセスを許可します。あるサービスで新しい機能を立ち上げる場合は、AWS は、追加された演算とリソースに対し、読み込み専用の権限を追加します。ジョブ機能ポリシーのリストと説明については、IAM ユーザーガイドのAWSジョブ関数のマネージドポリシーを参照してください。

AWS マネージドポリシー: CloudWatchNetworkMonitorServiceRolePolicy

`CloudWatchNetworkMonitorServiceRolePolicy` は、ユーザーに代わってアクションを実行することや、CloudWatch Network Monitor に関連付けられているリソースにアクセスすることをサービスに許可する、サービスリンクロールにアタッチされます。このポリシーは IAM アイデンティティにはアタッチできません。詳細については、「the section called “サービスリンクロール”」を参照してください。

CloudWatch Network Monitor の AWS マネージドポリシーへの更新

CloudWatch Network Monitor の AWS マネージドポリシーの更新に関する詳細を、このサービスがこれらの変更の追跡を開始した 2023 年 11 月以降の分について表示します。

変更	説明	日付
CloudWatchNetworkMonitorServiceRolePolicy: 新しいポリシー。	CloudWatch Network Monitor に追加された新しいポリシー。	2023 年 11 月 27 日
the section called “AWSServiceRoleForNetworkMonitor ”。新しいロール。	CloudWatch Network Monitor に追加された新しいロール。	2023 年 11 月 27 日

CloudWatch Network Monitor 用の IAM 許可

Amazon CloudWatch Network Monitor を使用するには、ユーザーに適切な許可が必要です。

Amazon CloudWatch のセキュリティの詳細については、「Amazon CloudWatch 用 Identity and Access Management」を参照してください。

モニタの表示に必要なアクセス許可

AWS Management Console で Amazon CloudWatch Network Monitor のモニタを表示するには、以下のアクセス許可が付与されたユーザーまたはロールとしてサインインする必要があります。

```
{
"Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "cloudwatch:GetMetricData",
                "networkmonitor:Get*",
                "networkmonitor:List*"
                ],
            "Resource": "*"
        }
    ]
}
```

モニターを作成するために必要なアクセス許可

Amazon CloudWatch Network Monitor でモニタを作成するには、ユーザーが Network Monitor に関連付けられたサービスリンクロールを作成するための許可を持っている必要があります。サービスリンクロールの詳細については、「CloudWatch Network Monitor のサービスリンクロールの使用」を参照してください。

AWS Management Console で Amazon CloudWatch Network Monitor のモニタを作成するには、以下のポリシーが含まれる許可を持つユーザーまたはロールとしてサインインする必要があります。

> **Note**
>
> 制限の厳しいアイデンティティベースの許可ポリシーを作成する場合、そのポリシーを持つユーザーはモニターを作成できません。

```
{
"Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "networkmonitor:*"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": "iam:CreateServiceLinkedRole",
            "Resource": "arn:aws:iam::*:role/aws-service-role/
networkmonitor.amazonaws.com/AWSServiceRoleForNetworkMonitor",
            "Condition": {
                "StringLike": {
                    "iam:AWSServiceName": "networkmonitor.amazonaws.com"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "iam:AttachRolePolicy",
                "iam:GetRole",
```

```
                "iam:PutRolePolicy"
            ],
            "Resource": "arn:aws:iam::*:role/aws-service-role/
networkmonitor.amazonaws.com/AWSServiceRoleForNetworkMonitor"
        },
        {
            "Action": [
                "ec2:CreateSecurityGroup",
                "ec2:CreateNetworkInterface",
                "ec2:CreateTags"
            ],
            "Effect": "Allow",
            "Resource": "*"
        }
    ]
}
```

CloudWatch Network Monitor のサービスリンクロールの使用

Amazon CloudWatch Network Monitor は、ユーザーに代わって AWS の他のサービスを呼び出すために必要なアクセス許可のために、次のサービスリンクロールを使用します。

- AWSServiceRoleForNetworkMonitor

AWSServiceRoleForNetworkMonitor

CloudWatch ネットワークモニタリングは、AWSServiceRoleForNetworkMonitor という名前のサービスリンクロールを使用して CloudWatch ネットワークモニタの更新と管理を行います。

AWSServiceRoleForNetworkMonitor サービスにリンクされたロールはその引き受け時に、以下のサービスを信頼します。

- networkmonitor.amazonaws.com

CloudWatchNetworkMonitorServiceRolePolicy はこのサービスリンクロールにアタッチされ、アカウント内の VPC や EC2 リソースにアクセスしたり、作成されたネットワークモニタを管理したりするためのアクセス権をサービスに付与します。

アクセス許可のグループ化

ポリシーは、以下のアクセス許可セットにグループ化されます。

- **cloudwatch** - ネットワークモニタリングメトリクスを CloudWatch リソースに公開することをサービスプリンシパルに許可します。
- **ec2** - モニタとプローブを作成または更新するために、アカウント内に VPC とサブネットを記述することをサービスプリンシパルに許可します。また、エンドポイントにモニタリングトラフィックを送信するモニタまたはプローブを設定するために、セキュリティグループ、ネットワークインターフェイス、およびそれらに関連する権限を作成、変更、削除することもサービスプリンシパルに許可します。

ポリシーの詳細については、「the section called “AWS マネージドポリシー”」を参照してください。

以下に CloudWatchNetworkMonitorServiceRolePolicy を示します。

```
{
 "Version": "2012-10-17",
 "Statement": [
  {
   "Sid": "PublishCw",
   "Effect": "Allow",
   "Action": "cloudwatch:PutMetricData",
   "Resource": "*",
   "Condition": {
    "StringEquals": {
     "cloudwatch:namespace": "AWS/NetworkMonitor"
    }
   }
  },
  {
   "Sid": "DescribeAny",
   "Effect": "Allow",
   "Action": [
    "ec2:DescribeNetworkInterfaces",
    "ec2:DescribeNetworkInterfaceAttribute",
    "ec2:DescribeVpcs",
    "ec2:DescribeNetworkInterfacePermissions",
    "ec2:DescribeSubnets",
    "ec2:DescribeSecurityGroups"
   ],
   "Resource": "*"
  },
  {
```

```
    "Sid": "DeleteModifyEc2Resources",
    "Effect": "Allow",
    "Action": [
     "ec2:AuthorizeSecurityGroupEgress",
     "ec2:CreateNetworkInterfacePermission",
     "ec2:DeleteNetworkInterfacePermission",
     "ec2:RevokeSecurityGroupEgress",
     "ec2:ModifyNetworkInterfaceAttribute",
     "ec2:DeleteNetworkInterface",
     "ec2:DeleteSecurityGroup"
    ],
    "Resource": [
     "arn:aws:ec2:*:*:network-interface/*",
     "arn:aws:ec2:*:*:security-group/*"
    ],
    "Condition": {
     "StringEquals": {
      "aws:ResourceTag/ManagedByCloudWatchNetworkMonitor": "true"
     }
    }
   }
  ]
}
```

サービスにリンクされたロールの作成

AWSServiceRoleForNetworkMonitor

AWSServiceRoleForNetworkMonitor ロールを手動で作成する必要はありません。

- AWSServiceRoleForNetworkMonitor ロールは、最初のネットワークモニタを作成するときに、CloudWatch Network Monitor によって作成されます。このロールは、それ以降に作成するすべてのモニタに適用されます。

お客様に代わってサービスリンクロールを作成するには、必要なアクセス許可がお客様に付与されていなければなりません。詳細については、IAM ユーザーガイドの「サービスにリンクされたロールのアクセス許可」を参照してください。

サービスにリンクされたロールを編集する

IAM を使用して AWSServiceRoleForNetworkMonitor の説明を編集することができます。詳細については、IAM ユーザーガイドの「サービスにリンクされたロールの編集」を参照してください。

サービスにリンクされたロールを削除する

CloudWatch Network Monitor を使用する必要がなくなった場合は、AWSServiceRoleForNetworkMonitor ロールを削除することをお勧めします。

これらのサービスリンクロールは、ネットワークモニタを削除した場合にのみ削除できます。ネットワークモニタの削除の詳細については、「Delete a network monitor」を参照してください。

サービスにリンクされたロールは、IAM コンソール、IAM CLI、または IAM API を使用して削除することができます。詳細については、IAM ユーザーガイドの「サービスにリンクされたロールの削除」を参照してください。

AWSServiceRoleForNetworkMonitor の削除後、新しいモニタを作成すると、CloudWatch Network Monitor によって再度このロールが作成されます。

CloudWatch Network Monitor のサービスリンクロールでサポートされるリージョン

CloudWatch Network Monitor は、そのサービスを利用できるすべての AWS リージョンで、サービスリンクロールをサポートします。詳細については、「AWS 全般のリファレンス」の「AWS エンドポイント」を参照してください。

サービスにリンクされたロールを削除する

CloudWatch Network Monitor を使用する必要がなくなった場合は、AWSServiceRoleForNetworkMonitor ロールを削除することをお勧めします。

これらのサービスリンクロールは、ネットワークモニタを削除した場合にのみ削除できます。ネットワークモニタの削除の詳細については、「Delete a network monitor」を参照してください。

サービスにリンクされたロールは、IAM コンソール、IAM CLI、または IAM API を使用して削除することができます。詳細については、IAM ユーザーガイドの「サービスにリンクされたロールの削除」を参照してください。

AWSServiceRoleForNetworkMonitor の削除後、新しいモニタを作成すると、CloudWatch Network Monitor によって再度このロールが作成されます。

料金

Amazon CloudWatch Network Monitor には前払いコストや長期間のコミットメントはありません。Network Monitor の料金には次の 2 つの要素があります。

- モニタリング対象リソースあたりの時間単位の料金、および
- CloudWatch メトリクス料金

ネットワークモニタを作成すると、モニタリング対象のリソースがネットワークモニタに関連付けられます。Network Monitor では、これらは Amazon Virtual Private Cloud (VPC) 内のサブネットになります。モニタリング対象リソースごとに、VPC 内の各サブネットから 4 つの宛先に最大 4 つのプローブを作成できます。請求額を抑えるには、モニタリング対象リソースの数を減らして、サブネットのカバレッジとオンプレミスの IP カバレッジを調整します。

料金の詳細については、「Amazon CloudWatch 料金表」を参照してください。

インフラストラクチャのモニタリング

このセクションのトピックでは、AWS リソースに関するオペレーションの可視性を高めるのに役立つ CloudWatch の機能について説明します。

トピック

Container Insights

CloudWatch Container Insights を使用して、コンテナ化されたアプリケーションとマイクロサービスのメトリクスとログを収集、集計、要約します。Container Insights は、Amazon Elastic Container Service (Amazon ECS)、Amazon Elastic Kubernetes Service (Amazon EKS)、および Amazon EC2 の Kubernetes プラットフォームで利用できます。Container Insights は、Amazon ECS と Amazon EKS の両方の AWS Fargate にデプロイされたクラスターからのメトリクスの収集をサポートしています。

CloudWatch は、CPU やメモリ、ディスク、ネットワークなど、多数のリソースのメトリクスを自動的に収集します。Container Insights では、問題の迅速な特定と解決に役立つ、コンテナの再起動失敗などの診断情報も提供されます。また、Container Insights が収集するメトリクスには CloudWatch アラームを設定できます。

Container Insights は、埋め込みメトリクス形式を使用して、パフォーマンスログイベントとしてデータを収集します。このパフォーマンスログイベントは、高濃度データを取り込み、大規模に保存することが可能な構造化された JSON スキーマを使用するエントリです。CloudWatch は、このデータから、クラスター、ノード、ポッド、タスク、サービスのレベルで CloudWatch メトリクスとして集約されたメトリクスを作成します。Container Insights が収集するメトリクスは、CloudWatch 自動ダッシュボードで使用でき、CloudWatch コンソールの [メトリクス] セクションでも表示できます。メトリクスは、コンテナタスクがしばらく実行されるまで表示されません。

Container Insights をデプロイする場合、パフォーマンスログイベント用のロググループが自動的に作成されます。このロググループを手動で作成する必要はありません。

Container Insights のコスト管理に役立てるため、CloudWatch により ログデータからあらゆるメトリクスが自動的に作成されるわけではありません。CloudWatch Logs Insights を使って生のパフォーマンスログイベントを分析すると、メトリクスと粒度レベルの詳細を確認できます。

Container Insights の元のバージョンでは、収集されたメトリクスおよび取り込まれたログはカスタムメトリクスとして課金されます。Amazon EKS 向けにオブザーバビリティが強化された Container Insights では、観察結果ごと Container Insights メトリクスおよびログに課金されます。保存されたメトリクスまたは取り込まれたログごとには課金されません。CloudWatch の料金の詳細については、Amazon CloudWatch の料金をご覧ください。

Amazon EKS および Kubernetes では、Container Insights はコンテナ化されたバージョンの CloudWatch エージェントを使用して、クラスターで実行中のすべてのコンテナを検出します。次に、パフォーマンススタックの各レイヤーでパフォーマンスデータを収集します。

Container Insights は、収集するログおよびメトリクスの AWS KMS key による暗号化をサポートします。この暗号化を有効にするには、Container Insights データを受信するロググループに対して AWS KMS 暗号化を手動で有効にする必要があります。これにより、Container Insights は提供された KMS キーを使用してこのデータを暗号化するようになります。対称 キーのみがサポートされます。ロググループの暗号化に非対称 KMS キーを使用しないでください。

詳細については、AWS KMS を使用した CloudWatch Logs でのログデータの暗号化を参照してください。

Amazon EKS 向けにオブザーバビリティが強化された Container Insights

2023 年 11 月 6 日、Container Insights の新しいバージョンがリリースされました。このバージョンでは、Amazon EC2 で実行されている Amazon EKS クラスター向けの強化されたオブザーバビリティがサポートされます。また、これらのクラスターから、より詳細なメトリクスを収集できます。インストール後は、Amazon EKS クラスターの詳細なインフラストラクチャテレメトリとコンテナログが自動的に収集されます。その後、キュレーションされたすぐに使用できるダッシュボードを使用して、アプリケーションおよびインフラストラクチャテレメトリを掘り下げて調べることができます。

Amazon EKS 向けにオブザーバビリティが強化された Container Insights では、コンテナレベルまでの詳細なヘルス、パフォーマンス、ステータスのメトリクスだけでなく、コントロールプレーンのメトリクスも収集できます。収集されるその他のメトリクスおよびディメンションの詳細については、「Amazon EKS および Kubernetes Container Insights のメトリクス」を参照してください。

2023 年 11 月 6 日以降に Amazon EC2 上の Amazon EKS クラスターで CloudWatch エージェントを使用して Container Insights をインストールした場合、Amazon EKS 向けにオブザーバビリティ

が強化された Container Insights を利用できます。そうでない場合、Amazon EKS 向けにオブザーバビリティが強化された Container Insights へのアップグレード の手順に従って Amazon EKS クラスターをこの新しいバージョンにアップグレードできます。

Container Insights は、CloudWatch クロスアカウントオブザーバビリティをサポートしています。単一のモニタリングアカウントで、単一のリージョン内の複数の AWS アカウントにまたがるアプリケーションをモニタリングし、トラブルシューティングできます。詳細については、「CloudWatch のクロスアカウントオブザーバビリティ」を参照してください。

Amazon EKS 向けにオブザーバビリティが強化された Container Insights は、Windows ワーカーノードもサポートします。

Amazon EKS 向けにオブザーバビリティが強化された Container Insights は、Fargate ではサポートされていません。

> **Note**
>
> Amazon EKS 向けにオブザーバビリティが強化された Container Insights にアップグレードできるクラスターがあるかどうかは、Container Insights コンソールに移動して確認できます。これを行うには、CloudWatch コンソールのナビゲーションペインで [インサイト]、[コンテナインサイト] の順に選択します。Container Insights コンソールには、アップグレード可能な Amazon EKS クラスターがあるかどうかを知らせるバナーと、アップグレードページへのリンクが表示されます。

サポートされているプラットフォーム

Container Insights は、Amazon Elastic Container Service、Amazon Elastic Kubernetes Service、および Amazon EC2 インスタンスの Kubernetes プラットフォームで利用できます。

- Amazon ECS の場合、Container Insights により Linux インスタンスおよび Windows Server インスタンスの両方のクラスター、タスク、およびサービスレベルでメトリクスが収集されます。インスタンスレベルでメトリクスを収集できるのは、Linux インスタンスのみです。

  Amazon ECS の場合、ネットワークメトリクスは `bridge` ネットワークモードおよび `awsvpc` ネットワークモードのコンテナでのみ使用できます。これらは、`host` ネットワークモードのコンテナでは使用できません。
- Amazon Elastic Kubernetes Service と、Amazon EC2 インスタンスの Kubernetes プラットフォームでは、Container Insights は Linux インスタンスでのみサポートされます。

CloudWatch エージェントコンテナイメージ

Amazon は、Amazon Elastic Container Registry に CloudWatch エージェントコンテナイメージを提供しています。詳細については、Amazon ECR の 「cloudwatch-agent」を参照してください。

サポートされているリージョン

Amazon ECS の Container Insights は、次のリージョンでサポートされています。

- 米国東部 (バージニア北部)
- 米国東部 (オハイオ)
- 米国西部 (北カリフォルニア)
- 米国西部 (オレゴン)
- アフリカ (ケープタウン)
- アジアパシフィック (香港)
- アジアパシフィック (ハイデラバード)
- アジアパシフィック (ジャカルタ)
- アジアパシフィック (ムンバイ)
- アジアパシフィック (大阪)
- アジアパシフィック (ソウル)
- アジアパシフィック (シンガポール)
- アジアパシフィック (東京)
- アジアパシフィック (シドニー)
- カナダ西部 (カルガリー)
- カナダ (中部)
- 欧州 (フランクフルト)
- 欧州 (アイルランド)
- 欧州 (ロンドン)
- 欧州 (ミラノ)
- 欧州 (パリ)
- 欧州 (スペイン)

- 欧州 (ストックホルム)
- 欧州 (チューリッヒ)
- 中東 (バーレーン)
- 中東 (アラブ首長国連邦)
- 南米 (サンパウロ)
- AWS GovCloud (米国東部)
- AWS GovCloud (米国西部)
- 中国 (北京)
- 中国 (寧夏)

Amazon EKS および Kubernetes でサポートされるリージョン

Amazon EKS および Kubernetes の Container Insights は、次のリージョンでサポートされています。

- 米国東部 (バージニア北部)
- 米国東部 (オハイオ)
- 米国西部 (北カリフォルニア)
- 米国西部 (オレゴン)
- アジアパシフィック (香港)
- アジアパシフィック (ムンバイ)
- アジアパシフィック (ソウル)
- アジアパシフィック (シンガポール)
- アジアパシフィック (シドニー)
- アジアパシフィック (東京)
- カナダ (中部)
- 中国 (北京)
- 中国 (寧夏)
- 欧州 (フランクフルト)
- 欧州 (アイルランド)
- 欧州 (ロンドン)

- 欧州 (パリ)
- ヨーロッパ (ストックホルム)
- 中東 (バーレーン)
- 南米 (サンパウロ)
- AWS GovCloud (米国東部)
- AWS GovCloud (米国西部)

Container Insights の設定

Container Insights のセットアッププロセスは Amazon ECS と Amazon EKS および Kubernetes では異なります。

トピック

- Amazon ECS での Container Insights のセットアップ
- Amazon EKS と Kubernetes での Container Insights のセットアップ

Amazon ECS での Container Insights のセットアップ

次のオプションのいずれかまたは両方を使用して、Amazon ECS クラスターで Container Insights を有効にできます。

- クラスターレベル、タスクレベル、およびサービスレベルのメトリクスの収集を開始するには、AWS Management Console または AWS CLI を使用します。
- CloudWatch エージェントをデーモンサービスとしてデプロイし、インスタンスでホストされているクラスターで Amazon EC2 インスタンスレベルのメトリクスの収集を開始します。

トピック

- クラスターおよびサービスレベルの Amazon ECS でメトリクスの Container Insights の設定
- AWS Distro for OpenTelemetry を使用した Amazon ECS での Container Insights の設定
- Amazon ECS で EC2 インスタンスレベルのメトリクスを収集するための CloudWatch エージェントのデプロイ
- Amazon ECS クラスター上で EC2 インスタンスレベルのメトリクスを収集するための AWS Distro for OpenTelemetry のデプロイ

- FireLens のログ送信先を CloudWatch Logs に設定する

クラスターおよびサービスレベルの Amazon ECS でメトリクスの Container Insights の設定

新規および既存の Amazon ECS クラスターで Container Insights を有効にできます。Container Insights は、クラスター、タスク、およびサービスレベルでメトリクスを収集します。Amazon ECS コンソールまたは AWS CLI のいずれかを使用して Container Insights を有効にすることができます。

Amazon EC2 インスタンスで Amazon ECS を使用している場合、Container Insights からネットワークおよびストレージメトリクスを収集するには、Amazon ECS エージェントバージョン 1.29 を含む AMI を使用してインスタンスを起動します。エージェントのバージョンを更新する方法については、「Amazon ECS コンテナエージェントの更新」を参照してください。

AWS CLI を使用して、アカウントレベルの許可を設定し、アカウント内に作成された新しい Amazon ECS クラスターに対して Container Insights を有効にすることができます。これを行うには、次のコマンドを入力します。

```
aws ecs put-account-setting --name "containerInsights" --value "enabled"
```

> **Note**
>
> Amazon ECS Container Insights メトリクスに使用するカスタマーマネージド AWS KMS キーが CloudWatch と連携するように設定されていない場合は、CloudWatch Logs で暗号化された口グを許可するようにキーポリシーを更新する必要があります。また、独自の AWS KMS キーを `/aws/ecs/containerinsights/ClusterName/performance` のロググループに関連付ける必要があります。詳細については、「Encrypt log data in CloudWatch Logs using AWS Key Management Service」を参照してください。

既存の Amazon ECS クラスターでの Container Insights のセットアップ

既存の Amazon ECS クラスターで Container Insights を有効にするには、次のコマンドを入力します。次のコマンドを使用するには、AWS CLI のバージョン 1.16.200 以降を実行している必要があります。

```
aws ecs update-cluster-settings --cluster myCICluster --settings
 name=containerInsights,value=enabled
```

新しい Amazon ECS クラスターでの Container Insights のセットアップ

新しい Amazon ECS クラスターで Container Insights を有効化する方法は 2 つあります。すべての新しいクラスターがデフォルトで Container Insights に対して有効になるように Amazon ECS を設定できます。それ以外の場合は、作成時に新しいクラスターを有効にすることができます。

AWS Management Console の使用

Container Insights は、デフォルトですべての新しいクラスターでオンにすることも、作成時に個々のクラスターでオンにすることもできます。

新しいクラスターで Container Insights をデフォルトでオンにするには

1. コンソールをhttps://console.aws.amazon.com/ecs/v2で開きます。
2. ナビゲーションペインで [Account Settings] (アカウント設定) を選択します。
3. [Update] (更新) を選択します。
4. クラスターで CloudWatch Container Insights をデフォルトで使用するには、[CloudWatch Container Insights] で、[CloudWatch Container Insights] を選択またはクリアします。
5. [Save changes] (変更の保存) をクリックします。

前述の手順を使用して、すべての新しいクラスターでデフォルトで Container Insights を有効にしていない場合は、次の手順を使用して、Container Insights が有効になっているクラスターを作成することができます。

Container Insights をオンにしてクラスターを作成するには

1. コンソール (https://console.aws.amazon.com/ecs/v2) を開きます。
2. ナビゲーションペインで [Clusters] (クラスター) を選択します。
3. [Clusters] (クラスター) ページで、[Create Cluster] (クラスターの作成) を選択します。
4. [Cluster configuration] (クラスター設定) で、[Cluster name] (クラスター名) に一意の名前を入力します。

   名前には、最大 255 文字 (大文字と小文字)、数字、およびハイフンを含めることができます。
5. Container Insights をオンにするには、[モニタリング] を展開し、[Container Insights の使用] をオンにします。

タスク定義を作成し、タスクを実行して、クラスター内でサービスを起動できるようになりました。詳細については、以下を参照してください。

- タスク定義の作成
- タスクの実行
- サービスの作成

AWS CLI を使用した新しい Amazon ECS クラスターでの Container Insights の設定

新しいクラスターでデフォルトで Container Insights を有効にするには、次のコマンドを入力します。

```
aws ecs put-account-setting --name "containerInsights" --value "enabled"
```

前述の手順を使用して、すべての新しいクラスターでデフォルトで Container Insights を有効にしない場合は、次のコマンドを入力して、Container Insights が有効になっている新しいクラスターを作成することができます。次のコマンドを使用するには、AWS CLI のバージョン 1.16.200 以降を実行している必要があります。

```
aws ecs create-cluster --cluster-name myCICluster --settings
 "name=containerInsights,value=enabled"
```

Amazon ECS クラスターでの Container Insights の無効化

既存の Amazon ECS クラスターで Container Insights を無効にするには、次のコマンドを入力します。

```
aws ecs update-cluster-settings --cluster myCICluster --settings
 name=containerInsights,value=disabled
```

AWS Distro for OpenTelemetry を使用した Amazon ECS での Container Insights の設定

このセクションは、AWS Distro for OpenTelemetry を使用して Amazon ECS クラスターで CloudWatch Container Insights を設定する場合に使用します。AWS Distro for Open Telemetry の詳細については、AWS Distro for OpenTelemetry を参照してください。

以下の手順では、クラスターが Amazon ECS を実行するように既に設定されていることを前提としています。Amazon ECS で AWS Distro for Open Telemetry を使用し、この目的のために Amazon

ECS クラスターを設定する方法の詳細については、Amazon Elastic Container Service で AWS Distro for OpenTelemetry Collector を設定するを参照してください。

ステップ 1: タスクロールを作成する

最初のステップでは、AWS OpenTelemetry Collector が使用するクラスター内にタスクロールを作成します。

AWS Distro for OpenTelemetry のタスクロールを作成するには

1. IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
2. ナビゲーションペインで、[Policies (ポリシー)] を選択し、[Create Policy (ポリシーの作成)] を選択します。
3. [JSON] タブを選択し、以下のポリシーをコピーします。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "logs:PutLogEvents",
                "logs:CreateLogGroup",
                "logs:CreateLogStream",
                "logs:DescribeLogStreams",
                "logs:DescribeLogGroups",
                "ssm:GetParameters"
            ],
            "Resource": "*"
        }
    ]
}
```

4. [Review policy (ポリシーの確認)] を選択します。
5. 名前に **AWSDistroOpenTelemetryPolicy** と入力し、[Create policy] (ポリシーの作成) を選択します。
6. 左側のナビゲーションペインから、[Roles] (ロール) を選択してから、[Create role] (ロールの作成) をクリックします。
7. サービスのリストで、[Elastic Container Service] を選択します。

8. ページの下部にある [Elastic Container Service Task] を選択し、[Next: Permissions] (次へ: アクセス許可) を選択します。
9. ポリシーのリストで AWSDistroOpenTelemetryPolicy を検索します。
10. AWSDistroOpenTelemetryPolicy の横にあるチェックボックスをオンにします。
11. [Next: Tags] (次へ: タグ)、[Next: Review] (次へ: 確認) の順に選択します。
12. [Role name] (ロール名) に **AWSOpenTelemetryTaskRole** と入力し、[Create role] (ロールの作成) を選択します。

ステップ 2: タスク実行ロールを作成する

次のステップでは、AWS OpenTelemetry Collector のタスク実行ロールを作成します。

AWS Distro for OpenTelemetry のタスク実行ロールを作成するには

1. IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
2. 左側のナビゲーションペインから、[Roles] (ロール) を選択してから、[Create role] (ロールの作成) をクリックします。
3. サービスのリストで、[Elastic Container Service] を選択します。
4. ページの下部にある [Elastic Container Service Task] を選択し、[Next: Permissions] (次へ: アクセス許可) を選択します。
5. ポリシーのリストで AmazonECSTaskExecutionRolePolicy を検索し、AmazonECSTaskExecutionRolePolicy の横にあるチェックボックスをオンにします。
6. ポリシーのリストで CloudWatchLogsFullAccess を検索し、CloudWatchLogsFullAccess の横にあるチェックボックスをオンにします。
7. ポリシーのリストで AmazonSSMReadOnlyAccess を検索し、AmazonSSMReadOnlyAccess の横にあるチェックボックスをオンにします。
8. [Next: Tags] (次へ: タグ)、[Next: Review] (次へ: 確認) の順に選択します。
9. [Role name] (ロール名) に **AWSOpenTelemetryTaskExecutionRole** と入力し、[Create role] (ロールの作成) を選択します。

ステップ 3: タスク定義を作成する

次のステップでは、タスク定義を作成します。

AWS Distro for OpenTelemetry のタスク定義を作成するには

1. コンソール (https://console.aws.amazon.com/ecs/v2) を開きます。
2. ナビゲーションペインで、[Task definitions] (タスク定義) を選択します。
3. [Create new task definition] (新しいタスク定義の作成)、[Create new task definition] (新しいタスク定義の作成) の順に選択します。
4. [Task definition family] (タスク定義ファミリー) を使用する場合、タスク定義ファミリーには、タスク定義に一意の名前を指定します。
5. コンテナを設定し、[次へ] を選択します。
6. [メトリクスとログ記録] で、[メトリクス収集の使用] を選択します。
7. [Next (次へ)] を選択します。
8. [Create] を選択します。

AWS OpenTelemetry Collector を Amazon ECS で使用する方法の詳細については、Amazon Elastic Container Service での AWS Distro for OpenTelemetry Collector の設定を参照してください。

ステップ 4: タスクを実行する

最後のステップでは、作成したタスクを実行します。

AWS Distro for OpenTelemetry のタスクを実行するには

1. コンソールをhttps://console.aws.amazon.com/ecs/v2で開きます。
2. 左のナビゲーションペインで、[Task Definitions] (タスク定義) を選択し、作成したタスクを選択します。
3. [アクション]、[デプロイ]、[タスクの実行] を選択します。
4. [Deploy] (デプロイ)、[Run task] (タスク実行) の順に選択します。
5. [コンピューティングオプション] セクションの [既存のクラスター] から、クラスターを選択します。
6. [Create] (作成) を選択します。
7. 次に、CloudWatch コンソールで新しいメトリクスを確認できます。
8. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
9. 左のナビゲーションペインで [Metrics] (メトリクス) を選択します。

ECS/ContainerInsights 名前空間が表示されます。その名前空間を選択すると、8 つのメトリクスが表示されます。

Amazon ECS で EC2 インスタンスレベルのメトリクスを収集するための CloudWatch エージェントのデプロイ

EC2 インスタンスでホストされている Amazon ECS クラスターからインスタンスレベルのメトリクスを収集するために CloudWatch エージェントをデプロイするには、デフォルト設定でクイックスタート設定を使用するか、エージェントを手動でインストールしてカスタマイズできるようにします。

どちらの方法でも、EC2 起動タイプでデプロイされた Amazon EC2 クラスターが既に少なくとも 1 つあることと、CloudWatch エージェントコンテナが Amazon EC2 インスタンスメタデータサービス (IMDS) にアクセスできることが必要になります。IMDS の詳細については、「インスタンスメタデータとユーザーデータ」を参照してください。

また、これらの方法では、AWS CLI がインストールされていることを前提としています。また、次の手順でコマンドを実行するには、[IAMFullAccess] ポリシーと [AmazonECS_FullAccess] ポリシーを持つアカウントまたはロールにログオンする必要があります。

トピック

- AWS CloudFormation を使用した高速セットアップ
- 手動およびカスタムセットアップ

AWS CloudFormation を使用した高速セットアップ

高速セットアップを使用するには、次のコマンドを入力して、エージェントをインストールする AWS CloudFormation を使用します。*cluster-name* と *cluster-regions* を Amazon ECS クラスターの名前とリージョンに置き換えます。

このコマンドは、IAM ロールの [CWAgentECSTaskRole] と [CWAgentECSExecutionRole] を作成します。これらのロールがアカウントにすでに存在する場合は、コマンドを入力ときに、`ParameterKey=CreateIAMRoles,ParameterValue=False` ではなく、`ParameterKey=CreateIAMRoles,ParameterValue=True` を使用します。そうしないと、コマンドは失敗します。

```
ClusterName=cluster-name
Region=cluster-region
```

```
curl -O https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-
insights/latest/ecs-task-definition-templates/deployment-mode/daemon-service/cwagent-
ecs-instance-metric/cloudformation-quickstart/cwagent-ecs-instance-metric-cfn.json
aws cloudformation create-stack --stack-name CWAgentECS-${ClusterName}-${Region} \
    --template-body file://cwagent-ecs-instance-metric-cfn.json \
    --parameters ParameterKey=ClusterName,ParameterValue=${ClusterName} \
                 ParameterKey=CreateIAMRoles,ParameterValue=True \
    --capabilities CAPABILITY_NAMED_IAM \
    --region ${Region}
```

(代替) 独自の IAM ロールの使用

[CWAgentECSTaskRole] および [CWAgentECSExecutionRole] ロールの代わりに、独自のカスタム ECS タスクロールと ECS タスク実行ロールを使用する場合は、最初に ECS タスクロールとして使用するロールに アタッチされている [CloudWatchAgentServerPolicy] が含まれていることを確認します。また、ECS タスク実行ロールとして使用するロールに、[CloudWatchAgentServerPolicy] ポリシー と [AmazonECSTaskExecutionRolePolicy] ポリシーの 両方がアタッチされていることを確認してください。次に、以下のコマンドを入力します。コマンドで、*task-role-arn* をカスタム ECS タスクロールの ARN に置き換え、*execution-role-arn* をカスタム ECS タスク実行ロールの ARN に置き換えます。

```
ClusterName=cluster-name
Region=cluster-region
TaskRoleArn=task-role-arn
ExecutionRoleArn=execution-role-arn
curl -O https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-
insights/latest/ecs-task-definition-templates/deployment-mode/daemon-service/cwagent-
ecs-instance-metric/cloudformation-quickstart/cwagent-ecs-instance-metric-cfn.json
aws cloudformation create-stack --stack-name CWAgentECS-${ClusterName}-${Region} \
    --template-body file://cwagent-ecs-instance-metric-cfn.json \
    --parameters ParameterKey=ClusterName,ParameterValue=${ClusterName} \
                 ParameterKey=TaskRoleArn,ParameterValue=${TaskRoleArn} \
                 ParameterKey=ExecutionRoleArn,ParameterValue=${ExecutionRoleArn} \
    --capabilities CAPABILITY_NAMED_IAM \
    --region ${Region}
```

高速セットアップのトラブルシューティング

AWS CloudFormation スタックのステータスを確認するには、次のコマンドを入力します。

```
ClusterName=cluster-name
```

```
Region=cluster-region
aws cloudformation describe-stacks --stack-name CWAgentECS-$ClusterName-$Region --
region $Region
```

StackStatus が CREATE_COMPLETEまたは CREATE_IN_PROGRESS 以外の場合、スタックイベントをチェックしてエラーを検出します。以下のコマンドを入力します。

```
ClusterName=cluster-name
Region=cluster-region
aws cloudformation describe-stack-events --stack-name CWAgentECS-$ClusterName-$Region
 --region $Region
```

cwagent デーモンサービスの状態を確認するには、次のコマンドを入力します。出力では、runningCount セクション のdesiredCount が deployment と等しいことがわかります。等しくない場合は、出力の failures セクションを確認してください。

```
ClusterName=cluster-name
Region=cluster-region
aws ecs describe-services --services cwagent-daemon-service --cluster $ClusterName --
region $Region
```

CloudWatch Logs コンソールを使用してエージェントログを確認することもできます。[/ecs/ecs-cwagent-daemon-service] ロググループを探します。

CloudWatch エージェントの AWS CloudFormation スタックの削除

AWS CloudFormation スタックを削除する必要がある場合は、次のコマンドを入力します。

```
ClusterName=cluster-name
Region=cluster-region
aws cloudformation delete-stack --stack-name CWAgentECS-${ClusterName}-${Region} --
region ${Region}
```

手動およびカスタムセットアップ

このセクションのステップに従って、CloudWatch エージェントを手動でデプロイして、EC2 インスタンスでホストされている Amazon ECS クラスターからインスタンスレベルのメトリクスを収集します。

必要な IAM ロールとポリシー

2 つの IAM ロールが必要です。まだ存在しない場合は、作成する必要があります。これらのロールの詳細については、「タスク用の IAM ロール」および「Amazon ECS タスク実行ロール」を参照してください。

- メトリクスを公開するために CloudWatch エージェントによって使用される、ECS タスクロール。このロールがすでに存在する場合は、CloudWatchAgentServerPolicy ポリシーがアタッチされていることを確認する必要があります。
- CloudWatch エージェントを起動するために Amazon ECS エージェントが使用する、EC2 タスク実行ロール。このロールがすでに存在する場合は、AmazonECSTaskExecutionRolePolicy ポリシー CloudWatchAgentServerPolicy とポリシーがアタッチされていることを確認する必要があります。

これらのロールがまだない場合は、次のコマンドを使用してロールを作成し、必要なポリシーをアタッチできます。この最初のコマンドは、ECS タスクロールを作成します。

```
aws iam create-role --role-name CWAgentECSTaskRole \
    --assume-role-policy-document "{\"Version\": \"2012-10-17\",\"Statement\": [{\"Sid
\": \"\",\"Effect\": \"Allow\",\"Principal\": {\"Service\": \"ecs-tasks.amazonaws.com
\"},\"Action\": \"sts:AssumeRole\"}]}"
```

前のコマンドを入力したら、コマンド出力の Arn の値を「TaskRoleArn」と書き留めます。このタスクは、後でタスク定義を作成するときに使用する必要があります。次に、以下のコマンドを入力して、必要なポリシーをアタッチします。

```
aws iam attach-role-policy --policy-arn arn:aws:iam::aws:policy/
CloudWatchAgentServerPolicy \
    --role-name CWAgentECSTaskRole
```

次のコマンドは、ECS タスク実行ロールを作成します。

```
aws iam create-role --role-name CWAgentECSExecutionRole \
    --assume-role-policy-document "{\"Version\": \"2012-10-17\",\"Statement\": [{\"Sid
\": \"\",\"Effect\": \"Allow\",\"Principal\": {\"Service\": \"ecs-tasks.amazonaws.com
\"},\"Action\": \"sts:AssumeRole\"}]}"
```

前のコマンドを入力したら、コマンド出力の Arn の値を「ExecutionRoleArn」と書き留めます。このタスクは、後でタスク定義を作成するときに使用する必要があります。次に、以下のコマンドを入力して、必要なポリシーをアタッチします。

```
aws iam attach-role-policy --policy-arn arn:aws:iam::aws:policy/
CloudWatchAgentServerPolicy \
    --role-name CWAgentECSExecutionRole

aws iam attach-role-policy --policy-arn arn:aws:iam::aws:policy/service-role/
AmazonECSTaskExecutionRolePolicy \
    --role-name CWAgentECSExecutionRole
```

タスク定義を作成し、デーモンサービスを起動する

タスク定義を作成し、それを使用して CloudWatch エージェントをデーモンサービスとして起動します。タスク定義を作成するには、次のコマンドを入力します。最初の行で、プレースホルダーをデプロイの実際の値に置き換えます。*logs-region* は、CloudWatch Logs が 配置されているリージョンで、*cluster-region* はクラスターが配置されているリージョンです。*task-role-arn* は、使用している ECS タスクロールの ARN で、*execution-role-arn* は ECS タスク実行ロールの ARN です。

```
TaskRoleArn=task-role-arn
ExecutionRoleArn=execution-role-arn
AWSLogsRegion=logs-region
Region=cluster-region
curl https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-
insights/latest/ecs-task-definition-templates/deployment-mode/daemon-service/cwagent-
ecs-instance-metric/cwagent-ecs-instance-metric.json \
    | sed "s|{{task-role-arn}}|${TaskRoleArn}|;s|{{execution-role-arn}}|
${ExecutionRoleArn}|;s|{{awslogs-region}}|${AWSLogsRegion}|" \
    | xargs -0 aws ecs register-task-definition --region ${Region} --cli-input-json
```

次に、以下のコマンドを実行してデーモンサービスを起動します。*cluster-name* と *cluster-regions* を Amazon ECS クラスターの名前とリージョンに置き換えます。

> **Important**
>
> このコマンドを実行する前に、すべてのキャパシティプロバイダー戦略を削除します。削除しない場合、コマンドは機能しません。

```
ClusterName=cluster-name
Region=cluster-region
aws ecs create-service \
    --cluster ${ClusterName} \
    --service-name cwagent-daemon-service \
    --task-definition ecs-cwagent-daemon-service \
    --scheduling-strategy DAEMON \
    --region ${Region}
```

次のエラーメッセージ、An error occurred (InvalidParameterException) when calling the CreateService operation: Creation of service was not idempotent が表示された場合は、cwagent-daemon-service という名前のデーモンサービスは既に作成されています。まず、次のコマンドを例として使用して、そのサービスを削除する必要があります。

```
ClusterName=cluster-name
Region=cluster-region
aws ecs delete-service \
    --cluster ${ClusterName} \
    --service cwagent-daemon-service \
    --region ${Region} \
    --force
```

(オプション) 詳細設定

オプションで、SSM を使用して、EC2 インスタンスでホストされている Amazon ECS クラスター内の CloudWatch エージェントの他の設定オプションを指定できます。オプションは以下のとおりです。

- metrics_collection_interval – CloudWatch エージェントがメトリクスを収集する頻度 (秒) 。デフォルトは 60 です。範囲は 1 ~ 172,000 です。
- endpoint_override – (オプション) ログの送信先となる別のエンドポイントを指定します。VPC のクラスターからデータを発行し、ログデータの送信先を VPC エンドポイントとする場合は、この操作を行うことをお勧めします。

  endpoint_override の値は URL 文字列であることが必要です。
- force_flush_interval – ログがサーバーに送信されるまでにメモリバッファ内に残留する最大時間を秒単位で指定します。このフィールドの設定にかかわらず、バッファ内のログのサイズが 1 MB に達すると、ログは即座にサーバーに送信されます。デフォルト値は 5 秒です。

- region – デフォルトでは、エージェントは Amazon ECS コンテナインスタンスがあるのと同じリージョンにメトリクスを発行します。これを上書きするには、ここで別のリージョンを指定します。例えば、"region" : "us-east-1"

次に、カスタマイズした設定の例を示します。

```
{
    "agent": {
        "region": "us-east-1"
    },
    "logs": {
        "metrics_collected": {
            "ecs": {
                "metrics_collection_interval": 30
            }
        },
        "force_flush_interval": 5
    }
}
```

Amazon ECS コンテナで CloudWatch エージェントの設定をカスタマイズするには

1. [AmazonSSMReadOnlyAccess] ポリシーが Amazon ECS タスク実行ロールにアタッチされていることを確認します。これを行うには、次のコマンドを入力します。この例では、Amazon ECS タスク実行ロールが CWAgentECSExecutionRole であることを前提としています。別のロールを使用している場合は、次のコマンドでそのロール名を置き換えます。

   ```
   aws iam attach-role-policy --policy-arn arn:aws:iam::aws:policy/
   AmazonSSMReadOnlyAccess \
           --role-name CWAgentECSExecutionRole
   ```

2. 前の例と同様のカスタマイズした設定ファイルを作成します。このファイル /tmp/ecs-cwagent-daemon-config.json の名前を変更します。
3. 次のコマンドを実行して、この設定を Parameter Store に配置します。*cluster-region* をお使いの Amazon ECS クラスターのリージョンに置き換えます。このコマンドを実行するには、AmazonSSMFullAccess ポリシーを持つユーザーまたはロールにログオンする必要があります。

   ```
   Region=cluster-region
   aws ssm put-parameter \
   ```

```
    --name "ecs-cwagent-daemon-service" \
    --type "String" \
    --value "`cat /tmp/ecs-cwagent-daemon-config.json`" \
    --region $Region
```

4. /tmp/cwagent-ecs-instance-metric.json などのタスク定義ファイルをローカルファイルにダウインロードします

```
curl https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-insights/latest/ecs-task-definition-templates/deployment-mode/daemon-service/cwagent-ecs-instance-metric/cwagent-ecs-instance-metric.json -o /tmp/cwagent-ecs-instance-metric.json
```

5. タスク定義ファイルを変更します。次のセクションを削除します。

```
"environment": [
                {
                    "name": "USE_DEFAULT_CONFIG",
                    "value": "True"
                }
            ],
```

そのセクションを次のように置き換えます。

```
"secrets": [
                {
                    "name": "CW_CONFIG_CONTENT",
                    "valueFrom": "ecs-cwagent-daemon-service"
                }
            ],
```

6. 次のステップに従って、エージェントをデーモンサービスとして再起動します。

   a. 次のコマンドを実行します。

```
TaskRoleArn=task-role-arn
ExecutionRoleArn=execution-role-arn
AWSLogsRegion=logs-region
Region=cluster-region
cat /tmp/cwagent-ecs-instance-metric.json \
    | sed "s|{{task-role-arn}}|${TaskRoleArn}|;s|{{execution-role-arn}}|
${ExecutionRoleArn}|;s|{{awslogs-region}}|${AWSLogsRegion}|" \
```

```
    | xargs -0 aws ecs register-task-definition --region ${Region} --cli-input-
json
```

b. 次のコマンドを実行して、デーモンサービスを起動します。*cluster-name* と *cluster-regions* を Amazon ECS クラスターの名前とリージョンに置き換えます。

```
ClusterName=cluster-name
Region=cluster-region
aws ecs create-service \
    --cluster ${ClusterName} \
    --service-name cwagent-daemon-service \
    --task-definition ecs-cwagent-daemon-service \
    --scheduling-strategy DAEMON \
    --region ${Region}
```

次のエラーメッセージ、`An error occurred (InvalidParameterException) when calling the CreateService operation: Creation of service was not idempotent` が表示された場合は、`cwagent-daemon-service` という名前のデーモンサービスは既に作成されています。まず、次のコマンドを例として使用して、そのサービスを削除する必要があります。

```
ClusterName=cluster-name
Region=Region
aws ecs delete-service \
    --cluster ${ClusterName} \
    --service cwagent-daemon-service \
    --region ${Region} \
    --force
```

Amazon ECS クラスター上で EC2 インスタンスレベルのメトリクスを収集するための AWS Distro for OpenTelemetry のデプロイ

このセクションの手順を使用して、AWS Distro for OpenTelemetry を使用して Amazon ECS クラスターで EC2 インスタンスレベルのメトリクスを収集します。AWS Distro for OpenTelemetry の詳細については、AWS Distro for OpenTelemetry を参照してください。

以下の手順では、クラスターが Amazon ECS を実行するように既に設定されていることを前提としています。このクラスターは、EC2 起動タイプを使用してデプロイする必要があります。Amazon ECS で AWS Distro for Open Telemetry を使用し、この目的のために Amazon ECS クラスターを設定する方法の詳細については、「Amazon Elastic Container Service で ECS EC2 インスタンスレベ

ルのメトリクスのための AWS Distro for OpenTelemetry Collector を設定する」を参照してください。

トピック

- AWS CloudFormation を使用した高速セットアップ
- 手動およびカスタムセットアップ

AWS CloudFormation を使用した高速セットアップ

Amazon ECS 向けの AWS Distro for OpenTelemetry コレクターを EC2 にインストールするための AWS CloudFormation テンプレートファイルをダウンロードします。次の curl コマンドを実行します。

```
curl -O https://raw.githubusercontent.com/aws-observability/aws-otel-collector/main/
deployment-template/ecs/aws-otel-ec2-instance-metrics-daemon-deployment-cfn.yaml
```

テンプレートファイルをダウンロードしたら、ファイルを開き、*PATH_TO_CloudFormation_TEMPLATE* をテンプレートファイルを保存したパスに置き換えます。そして、次のコマンドのように、以下のパラメータをエクスポートして、AWS CloudFormation コマンドを実行します。

- Cluster_Name — Amazon ECS のクラスター名
- AWS_REGION — データが送信されるリージョン
- PATH_TO_CloudFormation_TEMPLATE — AWS CloudFormation テンプレートファイルを保存したパス
- command — AWS Distro for Open Telemetry コレクターが Amazon EC2 上で Amazon ECS のインスタンスレベルのメトリクスを収集できるようにするには、このパラメータの `--config=/etc/ecs/otel-instance-metrics-config.yaml` を指定する必要があります。

```
ClusterName=Cluster_Name
Region=AWS_Region
command=--config=/etc/ecs/otel-instance-metrics-config.yaml
aws cloudformation create-stack --stack-name AOCECS-${ClusterName}-${Region} \
--template-body file://PATH_TO_CloudFormation_TEMPLATE \
--parameters ParameterKey=ClusterName,ParameterValue=${ClusterName} \
ParameterKey=CreateIAMRoles,ParameterValue=True \
ParameterKey=command,ParameterValue=${command} \
```

```
--capabilities CAPABILITY_NAMED_IAM \
--region ${Region}
```

このコマンドを実行したら、Amazon ECS コンソールを使用して、タスクが実行されているかどうかを確認します。

高速セットアップのトラブルシューティング

AWS CloudFormation スタックのステータスを確認するには、次のコマンドを入力します。

```
ClusterName=cluster-name
Region=cluster-region
aws cloudformation describe-stack --stack-name AOCECS-$ClusterName-$Region --region
 $Region
```

StackStatus が CREATE_COMPLETE または CREATE_IN_PROGRESS 以外の場合、スタックイベントをチェックしてエラーを検出します。次のコマンドを入力します。

```
ClusterName=cluster-name
Region=cluster-region
aws cloudformation describe-stack-events --stack-name AOCECS-$ClusterName-$Region --
region $Region
```

AOCECS デーモンサービスの状態を確認するには、次のコマンドを入力します。出力では、デプロイセクションの runningCount が desiredCount と等しいことがわかります。等しくない場合は、出力の「エラー」セクションを確認してください。

```
ClusterName=cluster-name
Region=cluster-region
aws ecs describe-services --services AOCECS-daemon-service --cluster $ClusterName --
region $Region
```

CloudWatch Logs コンソールを使用してエージェントログを確認することもできます。/aws/ecs/containerinsights/{ClusterName}/performance のロググループを探します。

手動およびカスタムセットアップ

このセクションの手順に従って、Amazon EC2 インスタンスでホストされている Amazon ECS クラスターからインスタンスレベルのメトリクスを収集するための AWS Distro for OpenTelemetry を手動でデプロイします。

ステップ 1: 必要なロールとポリシー

2 つの IAM ロールが必要です。まだ存在しない場合は、作成する必要があります。これらのロールの詳細については、「IAM ポリシーの作成」および「IAM ロールの作成」を参照してください。

ステップ 2: タスク定義を作成する

タスク定義を作成し、それを使用して AWS Distro for OpenTelemetry をデーモンサービスとして起動します。

タスク定義テンプレートを使用してタスク定義を作成するには、「AWS OTel Collector を使用して EC2 インスタンスの ECS EC2 タスク定義を作成する」の手順に従います。

Amazon ECS コンソールを使用してタスク定義を作成するには、「Amazon ECS の EC2 インスタンスメトリクス用に、AWS コンソールでタスク定義を作成して AWS OTel Collector をインストールする」の手順に従います。

ステップ 3: デーモンサービスを起動する

AWS Distro for OpenTelemetry をデーモンサービスとして起動するには、「デーモンサービスを使用して Amazon Elastic Container Service (Amazon ECS) でタスクを実行する」の手順に従います。

(オプション) 詳細設定

オプションで、SSM を使用して、Amazon EC2 インスタンスでホストされている Amazon ECS クラスター内の AWS Distro for OpenTelemetry の他の設定オプションを指定できます。設定ファイルの作成についての詳細は、「OpenTelemetry のカスタム設定」を参照してください。設定ファイルで使用できるオプションの詳細については、「AWS Container Insights Receiver」を参照してください。

FireLens のログ送信先を CloudWatch Logs に設定する

Amazon ECS 用の FireLens では、タスク定義パラメータを使用して、ログのストレージと分析のために Amazon CloudWatch Logs にログをルーティングできます。FireLens は Fluent Bit および Fluentd で動作します。提供されている AWS for Fluent Bit イメージを使用することも、独自の Fluent Bit または Fluentd イメージを使用することもできます。FireLens 設定を使用した Amazon ECS タスク定義の作成は、AWS SDK、AWS CLI、および AWS Management Console でサポートされています。CloudWatch の詳細については、「CloudWatch Logs とは」を参照してください。

Amazon ECS に FireLens を使用する場合、重要な考慮事項があります。詳細については、「考慮事項」をご参照ください。

Fluent Bit イメージ用の AWS を見つけるには、「Fluent Bit イメージのために AWS を使用する」をご参照ください。

FireLens 設定を使用するタスク定義を作成するには、「FireLens 設定を使用するタスク定義の作成」をご参照ください。

例

次のタスク定義の例は、CloudWatch Logs ロググループにログを転送するログ設定を指定する方法を示しています。詳細については、Amazon CloudWatch Logs ユーザーガイドの「Amazon CloudWatch Logs とは」を参照してください。

ログ設定オプションで、ロググループ名とロググループが存在するリージョンを指定します。Fluent Bit で自動的にロググループを作成するには、"auto_create_group":"true" を指定します。フィルタリングに役立つログストリームプレフィックスとしてタスク ID を指定することもできます。詳細については、「CloudWatch Logs 用の Fluent Bit プラグイン」を参照してください。

```
{
 "family": "firelens-example-cloudwatch",
 "taskRoleArn": "arn:aws:iam::123456789012:role/ecs_task_iam_role",
 "containerDefinitions": [
  {
   "essential": true,
   "image": "906394416424.dkr.ecr.us-west-2.amazonaws.com/aws-for-fluent-bit:latest",
   "name": "log_router",
   "firelensConfiguration": {
    "type": "fluentbit"
   },
   "logConfiguration": {
    "logDriver": "awslogs",
    "options": {
     "awslogs-group": "firelens-container",
     "awslogs-region": "us-west-2",
     "awslogs-create-group": "true",
     "awslogs-stream-prefix": "firelens"
    }
   },
   "memoryReservation": 50
   },
   {
    "essential": true,
    "image": "nginx",
    "name": "app",
```

```
    "logConfiguration": {
     "logDriver":"awsfirelens",
     "options": {
     "Name": "cloudwatch",
     "region": "us-west-2",
     "log_key": "log",
                                  "log_group_name": "/aws/ecs/containerinsights/
$(ecs_cluster)/application",
     "auto_create_group": "true",
     "log_stream_name": "$(ecs_task_id)"
    }
   },
   "memoryReservation": 100
  }
 ]
}
```

Amazon EKS と Kubernetes での Container Insights のセットアップ

Container Insights は、Amazon EKS バージョン 1.23 以降でサポートされています。インストールのクイックスタート方法は、バージョン 1.24 以降でのみサポートされています。

Amazon EKS または Kubernetes で Container Insights をセットアップするための全体的なプロセスを次に示します。

1. 必要な前提条件を満たしていることを確認します。
2. Amazon CloudWatch Observability EKS アドオン、CloudWatch エージェント、または AWS Distro for OpenTelemetry をクラスターにセットアップして、メトリクスを CloudWatch に送信します。

> **Note**
>
> Amazon EKS 向けにオブザーバビリティが強化された Container Insights を使用するには、Amazon CloudWatch Observability EKS アドオンまたは CloudWatch エージェントを使用する必要があります。Container Insights のこのバージョンについての詳細は、「Amazon EKS 向けにオブザーバビリティが強化された Container Insights」を参照してください。
>
> Fargate で Container Insights を使用するには、AWS Distro for OpenTelemetry を使用する必要があります。Amazon EKS 向けにオブザーバビリティが強化された Container Insights は、Fargate ではサポートされていません。

> **Note**
>
> Container Insights が Amazon EKS クラスターの Windows ワーカーノードをサポートするようになりました。Amazon EKS 向けにオブザーバビリティが強化された Container Insights も、Windows でサポートされています。Windows で Container Insights を有効にする詳細については、「Container Insights の強化されたオブザーバビリティを有効にした状態で CloudWatch エージェントを使用する」を参照してください。

Fluent Bit または Fluentd をセットアップしてログを CloudWatch Logs に送信します。(Amazon CloudWatch Observability EKS アドオンをインストールする場合、これはデフォルトで有効になっています。)

CloudWatch エージェントを使用している場合、これらのステップは、クイックスタートセットアップの一部として一度に実行することも、個別に行うこともできます。

3. (オプション) Amazon EKS コントロールプレーンのログ記録の設定
4. (オプション) クラスターで StatsD メトリクスを CloudWatch に送信するように StatsD エンドポイントとして CloudWatch エージェントをセットアップします。
5. (オプション) App Mesh Envoy アクセスログを有効にします。

Container Insights の元のバージョンでは、収集されたメトリクスおよび取り込まれたログはカスタムメトリクスとして課金されます。Amazon EKS 向けにオブザーバビリティが強化された Container Insights では、観察結果ごと Container Insights メトリクスおよびログに課金されます。保存されたメトリクスまたは取り込まれたログごとには課金されません。CloudWatch の料金の詳細については、Amazon CloudWatch の料金をご覧ください。

トピック

- 前提条件を確認する
- Container Insights の強化されたオブザーバビリティを有効にした状態で CloudWatch エージェントを使用する
- AWS Distro for OpenTelemetry の使用
- CloudWatch Logs にログを送信する
- Amazon EKS および Kubernetes での Container Insights の更新または削除

前提条件を確認する

Amazon EKS または Kubernetes で Container Insights をインストールする前に、以下を確認してください。これらの前提条件は、CloudWatch エージェントまたは AWS Distro for OpenTelemetry を使用して Amazon EKS クラスターに Container Insights をセットアップする場合に適用されます。

- Amazon EKS および Kubernetes に Container Insights がサポートされているいずれかのリージョンに、ノードがアタッチされて機能している Amazon EKS または Kubernetes クラスターがある。サポートされているリージョンのリストについては、「Container Insights」を参照してください。
- `kubectl` がインストールされ、実行されている。詳細については、Amazon EKS ユーザーガイドの「`kubectl` のインストール」を参照してください。
- Amazon EKS を使用する代わりに、AWS で実行されている Kubernetes を使用している場合は、以下の前提条件も必要になります。
  - Kubernetes クラスターで、ロールベースのアクセス制御 (RBAC) が有効になっている。詳細については、Kubernetes Reference の RBAC 認証の使用を参照してください。
  - kubelet で Webhook 認証モードが有効になっている。詳細については、Kubernetes Reference の Kubelet の認証/認可を参照してください。

また、Amazon EKS ワーカーノードが CloudWatch にメトリクスとログを送信できるようにするには、IAM にもアクセス許可を付与する必要があります。次の 2 通りの方法があります。

- ワーカーノードの IAM ロールにポリシーをアタッチします。これは、Amazon EKS クラスターと他の Kubernetes クラスターの両方に有効です。
- クラスターでサービスアカウントの IAM ロールを使用し、このロールにポリシーをアタッチします。これは Amazon EKS クラスターでのみ機能します。

最初のオプションはノード全体へのアクセス許可を CloudWatch に付与しますが、サービスアカウントの IAM ロールを使用した場合は、該当する daemonset ポッドへのアクセス権のみを CloudWatch に付与します。

ワーカーノードの IAM ロールにポリシーをアタッチする

ワーカーノードの IAM ロールにポリシーをアタッチするには、次の手順に従います。これは、Amazon EKS クラスターと、Amazon EKS 外の Kubernetes クラスターの両方で機能します。

ワーカーノードの IAM ロールに必要なポリシーをアタッチするには

1. Amazon EC2 コンソール (https://console.aws.amazon.com/ec2/) を開きます。
2. ワーカーノードのインスタンスの 1 つを選択し、説明で IAM ロールを選択します。
3. IAM ロールページで、[Attach policies (ポリシーをアタッチします)] を選択します。
4. ポリシーのリストで、[CloudWatchAgentServerPolicy] の横にあるチェックボックスを選択します。必要に応じて、検索ボックスを使用してポリシーを見つけます。
5. [ポリシーのアタッチ] を選択します。

Amazon EKS 外で Kubernetes クラスターを実行している場合、まだワーカーノードにアタッチされた IAM ロールがない可能性があります。アタッチしていない場合は、最初に IAM ロールをインスタンスにアタッチし、前の手順で説明しているようにポリシーを追加する必要があります。インスタンスにロールをアタッチする方法の詳細については、Windows インスタンス用 Amazon EC2 ユーザーガイドの「IAM ロールをインスタンスにアタッチする」を参照してください。

Amazon EKS 外で Kubernetes クラスターを実行しており、メトリクスで EBS ボリューム ID を収集する場合は、インスタンスにアタッチされた IAM ロールに別のポリシーを追加する必要があります。以下をインラインポリシーとして追加します。詳細については、IAM ユーザーガイドの「IAM ID アクセス許可の追加と削除」を参照してください。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "ec2:DescribeVolumes"
            ],
            "Resource": "*",
            "Effect": "Allow"
        }
    ]
}
```

IAM サービスアカウントのロールを使用する

この方法は Amazon EKS クラスターでのみ機能します。

IAM サービスアカウントロールを使用して CloudWatch にアクセス許可を付与するには

1. クラスターでサービスアカウントの IAM ロールを有効にします (まだ有効にしていない場合)。詳細については、「クラスターでサービスアカウントの IAM ロールを有効にする」を参照してください。
2. まだ設定していない場合は、サービスアカウントが IAM ロールを使用するように設定します。詳細については、「IAM ロールを引き受けるための Kubernetes サービスアカウントの設定」を参照してください。

   ロールの作成時に、ロール用に作成するポリシーに加えて、CloudWatchAgentServerPolicy IAM ポリシーをロールにアタッチします。また、このロールにリンクされている関連する Kubernetes サービスアカウントは、次のステップで CloudWatch と Fluent Bit のデーモンセットがデプロイされる `amazon-cloudwatch` 名前空間に作成する必要があります
3. IAM ロールをクラスターのサービスアカウントに関連付けます (まだ関連付けていない場合)。詳細については、「IAM ロールを引き受けるための Kubernetes サービスアカウントの設定」を参照してください。

Container Insights の強化されたオブザーバビリティを有効にした状態で CloudWatch エージェントを使用する

次のセクションのいずれかに記載されている手順を実施して、CloudWatch エージェントを使用して Amazon EKS クラスターまたは Kubernetes クラスターに Container Insights をセットアップします。クイックスタートの手順は、Amazon EKS バージョン 1.24 以降でのみサポートされています。

> **Note**
>
> Container Insights は、次のセクションのいずれかの手順に従いインストールできます。3 つの手順すべてを実施する必要はありません。

トピック

- Amazon CloudWatch Observability EKS アドオンのインストール
- Amazon EKS および Kubernetes の Container Insights のクイックスタートセットアップ
- クラスターメトリクスを収集するよう CloudWatch エージェントをセットアップする

Amazon CloudWatch Observability EKS アドオンのインストール

Amazon EKS アドオンを使用して、Amazon EKS 向けにオブザーバビリティが強化された Container Insights をインストールできます。このアドオンは、CloudWatch エージェントをインストールしてクラスターからインフラストラクチャメトリクスを送信します。また、Fluent Bit をインストールしてコンテナログを送信するほか、CloudWatch Application Signals を有効にしてアプリケーションパフォーマンステレメトリを送信します。

Amazon EKS アドオンバージョン 1.5.0 以降を使用すると、Container Insights はクラスター内の Linux と Windows の両方のワーカーノードで有効になります。現在、Amazon EKS の Windows では、Application Signals はサポートされていません。

Amazon EKS アドオンは、Amazon EKS ではなく Kubernetes を実行しているクラスターではサポートされていません。

Amazon CloudWatch Observability EKS アドオンについての詳細は、「Amazon CloudWatch Observability EKS アドオンを使用して CloudWatch エージェントをインストールする」を参照してください。

Amazon CloudWatch Observability EKS アドオンをインストールするには

1. まず、CloudWatchAgentServerPolicy IAM ポリシーをワーカーノードにアタッチして、必要なアクセス許可を設定します。これを行うには、次のコマンドを入力します。*my-worker-node-role* を Kubernetes ワーカーノードが使用する IAM ロールに置き換えます。

   ```
   aws iam attach-role-policy \
   --role-name my-worker-node-role \
   --policy-arn arn:aws:iam::aws:policy/CloudWatchAgentServerPolicy
   ```

2. 次のコマンドを入力してアドオンをインストールします。

   ```
   aws eks create-addon --cluster-name my-cluster-name --addon-name amazon-cloudwatch-observability
   ```

Amazon EKS および Kubernetes の Container Insights のクイックスタートセットアップ

> **Important**
>
> Amazon EKS クラスターに Container Insights をインストールする場合は、このセクションの手順を使用するのではなく Amazon CloudWatch Observability EKS アドオンを使用してイ

ンストールすることをお勧めします。また、高速コンピューティングネットワークを取得するには、Amazon CloudWatch Observability EKS アドオンを使用する必要があります。詳細と手順については、「Amazon CloudWatch Observability EKS アドオンのインストール」を参照してください。

Container Insights のセットアップを完了するには、このセクションのクイックスタート手順に従います。2023 年 11 月 6 日以降にこのセクションの手順を使用して Amazon EKS クラスターにインストールを行う場合は、Amazon EKS 向けにオブザーバビリティが強化された Container Insights をそのクラスターにインストールします。

Important

このセクションに記載されている手順を完了する前に、IAM アクセス許可を含む前提条件を確認しておく必要があります。詳細については、「前提条件を確認する」を参照してください。

または、代わりに「クラスターメトリクスを収集するよう CloudWatch エージェントをセットアップする」および「CloudWatch Logs にログを送信する」の 2 つのセクションの手順に従うことができます。これらのセクションでは、CloudWatch エージェントがどのように Amazon EKS および Kubernetes と動作するかについてより詳しく説明されていますが、追加のインストール手順を実行する必要があります。

Container Insights の元のバージョンでは、収集されたメトリクスおよび取り込まれたログはカスタムメトリクスとして課金されます。Amazon EKS 向けにオブザーバビリティが強化された Container Insights では、観察結果ごと Container Insights メトリクスおよびログに課金されます。保存されたメトリクスまたは取り込まれたログごとには課金されません。CloudWatch の料金の詳細については、Amazon CloudWatch の料金をご覧ください。

Note

Amazon は、Container Insights のデフォルトのログソリューションとして Fluent Bit の提供を開始しました。これにより、パフォーマンスの大幅な向上が見込めます。Fluentd の代わりに Fluent Bit を使用することをお勧めします。

CloudWatch エージェントオペレーターと Fluent Bit を使用したクイックスタート

Fluent Bit については、最適化されたバージョンと Fluentd に似た使用感を提供するバージョンの 2 つの設定があります。クイックスタート設定では、最適化されたバージョンが使用されます。Fluentd 互換の設定の詳細については、「CloudWatch Logs へログを送信する DaemonSet として Fluent Bit を設定する」を参照してください。

CloudWatch エージェントオペレーターは、Amazon EKS クラスターにインストールされる追加のコンテナです。OpenTelemetry Operator for Kubernetes をモデルにしています。オペレーターは、クラスター内の Kubernetes リソースのライフサイクルを管理します。CloudWatch エージェント、DCGM Exporter (NVIDIA)、および AWS Neuron Monitor を Amazon EKS クラスターにインストールして管理します。Fluent Bit および Windows 用 CloudWatch エージェントは、オペレーターが管理しなくても Amazon EKS クラスターに直接インストールされます。

安全性の高い多機能な証明機関を利用できるように、CloudWatch エージェントオペレーターには cert-manager が必要です。これは、Kubernetes で TLS 証明書を管理するのに広く採用されているソリューションです。cert-manager を使用すると、これらの証明書を取得、更新、管理、使用するプロセスを簡素化できます。これにより、証明書が有効で最新の状態であることが保証されます。有効期限前の設定しておいた時刻になると、証明書が更新されます。また、AWS Certificate Manager Private Certificate Authority などサポートされているさまざまな発行元から簡単に証明書を発行できます。

クイックスタートを使用して Container Insights をデプロイするには

1. まだクラスターにインストールされていない場合は、cert-manager をインストールします。詳細については、cert-manager Installation を参照してください。
2. 次のコマンドを入力して、カスタムリソース定義 (CRD) をインストールします。

```
curl https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-
insights/main/k8s-quickstart/cwagent-custom-resource-definitions.yaml | kubectl
 apply --server-side -f -
```

3. 次のコマンドを入力して、オペレーターをインストールします。*my-cluster-name* は Amazon EKS または Kubernetes クラスターの名前に置き換え、*my-cluster-region* はログが発行されるリージョンの名前に置き換えます。AWS アウトバウンドデータ転送コストを削減するために、クラスターがデプロイされているのと同じリージョンを使用することをお勧めします。

```
ClusterName=my-cluster-name
```

```
RegionName=my-cluster-region
curl https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-
container-insights/main/k8s-quickstart/cwagent-operator-rendered.yaml | sed 's/
{{cluster_name}}/'${ClusterName}'/g;s/{{region_name}}/'${RegionName}'/g' | kubectl
 apply -f -
```

例えば、MyCluster という名前のクラスターに Container Insights をデプロイし、ログとメトリクスを米国西部 (オレゴン) に発行するには、次のコマンドを入力します。

```
ClusterName='MyCluster'
RegionName='us-west-2'
curl https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-
container-insights/main/k8s-quickstart/cwagent-operator-rendered.yaml | sed 's/
{{cluster_name}}/'${ClusterName}'/g;s/{{region_name}}/'${RegionName}'/g' | kubectl
 apply -f -
```

Container Insights からの移行

Amazon EKS クラスターで既に Container Insights を設定しており、Amazon EKS 向けにオブザーバビリティが強化された Container Insights に移行する場合は、「Amazon EKS 向けにオブザーバビリティが強化された Container Insights へのアップグレード」を参照してください。

Container Insights の使用

クイックスタートセットアップの使用後に Container Insights を削除する場合は、次のコマンドを入力します。

```
ClusterName=my-cluster-name
RegionName=my-cluster-region
curl https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-
container-insights/main/k8s-quickstart/cwagent-operator-rendered.yaml | sed 's/
{{cluster_name}}/'${ClusterName}'/g;s/{{region_name}}/'${RegionName}'/g' | kubectl
 delete -f -
curl https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-
insights/main/k8s-quickstart/cwagent-custom-resource-definitions.yaml | kubectl delete
 -f -
```

クラスターメトリクスを収集するよう CloudWatch エージェントをセットアップする

> **Important**
>
> Amazon EKS クラスターに Container Insights をインストールする場合は、このセクションの手順を使用するのではなく Amazon CloudWatch Observability EKS アドオンを使用してインストールすることをお勧めします。詳細と手順については、「Amazon CloudWatch Observability EKS アドオンのインストール」を参照してください。

Container Insights をセットアップしてメトリクスを収集するには、「Amazon EKS および Kubernetes の Container Insights のクイックスタートセットアップ」の手順に従うか、このセクションの手順に従います。以下のステップでは、クラスターからメトリクスを収集できるよう CloudWatch エージェントをセットアップします。

2023 年 11 月 6 日以降にこのセクションの手順を使用して Amazon EKS クラスターにインストールを行う場合は、Amazon EKS 向けにオブザーバビリティが強化された Container Insights をそのクラスターにインストールします。

ステップ 1: CloudWatch の名前空間を作成する

CloudWatch に対して `amazon-cloudwatch` という Kubernetes 名前空間を作成するには、次のステップを使用します。すでにこの名前空間を作成している場合は、以下のステップをスキップできます。

CloudWatch の名前空間を作成するには

- 以下のコマンドを入力します。

```
kubectl apply -f https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-
container-insights/latest/k8s-deployment-manifest-templates/deployment-mode/
daemonset/container-insights-monitoring/cloudwatch-namespace.yaml
```

ステップ 2: クラスターのサービスアカウントを作成する

次のステップを使用して、CloudWatch エージェントのサービスアカウントを作成します (作成済みでない場合)。

CloudWatch エージェントのサービスアカウントを作成するには

- 以下のコマンドを入力します。

```
kubectl apply -f https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-
container-insights/latest/k8s-deployment-manifest-templates/deployment-mode/
daemonset/container-insights-monitoring/cwagent/cwagent-serviceaccount.yaml
```

前のステップに従っていないが、使用する CloudWatch エージェントのサービスアカウントが既にある場合は、そのアカウントに次のルールがあることを確認する必要があります。さらに、Container Insights のインストールの残りの手順では、`cloudwatch-agent` の代わりにそのサービスアカウントの名前を使用する必要があります。

```
rules:
  - apiGroups: [""]
    resources: ["pods", "nodes", "endpoints"]
    verbs: ["list", "watch"]
  - apiGroups: [ "" ]
    resources: [ "services" ]
    verbs: [ "list", "watch" ]
  - apiGroups: ["apps"]
    resources: ["replicasets", "daemonsets", "deployments", "statefulsets"]
    verbs: ["list", "watch"]
  - apiGroups: ["batch"]
    resources: ["jobs"]
    verbs: ["list", "watch"]
  - apiGroups: [""]
    resources: ["nodes/proxy"]
    verbs: ["get"]
  - apiGroups: [""]
    resources: ["nodes/stats", "configmaps", "events"]
    verbs: ["create", "get"]
  - apiGroups: [""]
    resources: ["configmaps"]
    resourceNames: ["cwagent-clusterleader"]
    verbs: ["get","update"]
  - nonResourceURLs: ["/metrics"]
    verbs: ["get", "list", "watch"]
```

ステップ 3: CloudWatch エージェント用の ConfigMap を作成する

CloudWatch エージェント用の ConfigMap を作成するには、次のステップを使用します。

CloudWatch エージェント用の ConfigMap を作成するには

1. 以下のコマンドを実行して、ConfigMap YAML を kubectl クライアントホストにダウンロードします。

```
curl -O https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-insights/latest/k8s-deployment-manifest-templates/deployment-mode/daemonset/container-insights-monitoring/cwagent/cwagent-configmap.yaml
```

2. ダウンロードした YAML ファイルを次のように編集します。

   - cluster_name – kubernetes セクションで、{{cluster_name}} をクラスターの名前と置き換えます。{{}} 文字を削除します。または、Amazon EKS クラスターを使用している場合、"cluster_name" フィールドと値を削除できます。そうした場合、CloudWatch エージェントは Amazon EC2 タグからクラスター名を検出します。

3. (オプション) モニターリング要件に基づいて、次のように ConfigMap をさらに変更します。

   - metrics_collection_interval – kubernetes セクションで、エージェントがメトリクスを収集する頻度を指定できます。デフォルト値は 60 秒です。kubelet でのデフォルトの cadvisor 収集間隔は 15 秒であるため、この値を 15 秒未満に設定しないでください。
   - endpoint_override – logs セクションで、デフォルトのエンドポイントを上書きする場合は、CloudWatch Logs エンドポイントを指定できます。VPC のクラスターからデータを発行し、データの送信先を VPC エンドポイントとする場合は、この操作を行うことをお勧めします。
   - force_flush_interval – logs セクションで、CloudWatch Logs に発行される前にバッチ処理のログイベントのインターバルを指定できます。デフォルト値は 5 秒です。
   - region – デフォルトでは、エージェントはワーカーノードがあるリージョンにメトリクスを発行します。これを無効にするには、region セクションに agent フィールドを追加できます。例: "region":"us-west-2"。
   - statsd セクション – クラスターの各ワーカーノードで StatsD リスナーとして CloudWatch Logs エージェントを実行する場合は、次の例のように statsd セクションに metrics セクションを追加できます。このセクションの他の StatsD オプションについては、「StatsD を使用してカスタムメトリクスを取得する 」を参照してください。

```
"metrics": {
  "metrics_collected": {
    "statsd": {
      "service_address":":8125"
    }
  }
}
```

JSON セクションの完全な例を次に示します。

```
{
    "agent": {
        "region": "us-east-1"
    },
    "logs": {
        "metrics_collected": {
            "kubernetes": {
                "cluster_name": "MyCluster",
                "metrics_collection_interval": 60
            }
        },
        "force_flush_interval": 5,
        "endpoint_override": "logs.us-east-1.amazonaws.com"
    },
    "metrics": {
        "metrics_collected": {
            "statsd": {
                "service_address": ":8125"
            }
        }
    }
}
```

4. 次のコマンドを実行して、クラスターに ConfigMap を作成します。

```
kubectl apply -f cwagent-configmap.yaml
```

ステップ 4: CloudWatch エージェントを DaemonSet としてデプロイする

CloudWatch エージェントのインストールを完了してコンテナメトリクスの収集を開始するには、次のステップを使用します。

CloudWatch エージェントを DaemonSet としてデプロイするには

1. • クラスターで StatsD を使用しない場合は、次のコマンドを入力します。

```
kubectl apply -f https://raw.githubusercontent.com/aws-samples/amazon-
cloudwatch-container-insights/latest/k8s-deployment-manifest-templates/
deployment-mode/daemonset/container-insights-monitoring/cwagent/cwagent-
daemonset.yaml
```

   • StatsD を使用する場合は、以下の手順に従います。

   a. 次のコマンドを実行して、kubectl クライアントに DaemonSet YAML をダウンロードします。

```
curl -O  https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-
container-insights/latest/k8s-deployment-manifest-templates/deployment-
mode/daemonset/container-insights-monitoring/cwagent/cwagent-daemonset.yaml
```

   b. 次のように、port ファイルの cwagent-daemonset.yaml セクションのコメントを解除します。

```
ports:
  - containerPort: 8125
    hostPort: 8125
    protocol: UDP
```

   c. 次のコマンドを実行して、クラスターに CloudWatch エージェントをデプロイします。

```
kubectl apply -f cwagent-daemonset.yaml
```

   d. 次のコマンドを実行して、クラスターの Windows ノードに CloudWatch エージェントをデプロイします。StatsD リスナーは Windows 上の CloudWatch エージェントではサポートされていません。

```
kubectl apply -f cwagent-daemonset-windows.yaml
```

2. 次のコマンドを実行して、エージェントがデプロイされたことを確認します。

```
kubectl get pods -n amazon-cloudwatch
```

操作を完了すると、CloudWatch エージェントは /aws/containerinsights/*Cluster_Name*/performance という名前のロググループを作成し、このロググループにパフォーマンスログイベントを送信します。また、このエージェントを StatsD リスナーとしてもセットアップする場合、エージェントはアプリケーションポッドがスケジュールされたノードの IP アドレスを使用して、ポート 8125 で StatsD メトリクスをリッスンします。

トラブルシューティング

エージェントが正しくデプロイされない場合は、次の操作を試してください。

- 次のコマンドを実行して、ポッドのリストを取得します。

```
kubectl get pods -n amazon-cloudwatch
```

- 次のコマンドを実行して、出力の下部にあるイベントを確認します。

```
kubectl describe pod pod-name -n amazon-cloudwatch
```

- 次のコマンドを実行して、ログを確認します。

```
kubectl logs pod-name  -n amazon-cloudwatch
```

AWS Distro for OpenTelemetry の使用

AWS Distro for OpenTelemetry コレクターを使用して、Amazon EKS クラスターからメトリクスを収集するように Container Insights を設定できます。AWS Distro for OpenTelemetry の詳細については、AWS Distro for OpenTelemetry を参照してください。

> **Important**
>
> AWS Distro for OpenTelemetry を使用してインストールする場合、Container Insights はインストールされますが、Amazon EKS 向けにオブザーバビリティが強化された Container Insights はインストールされません。また、Amazon EKS 向けにオブザーバビリティが強化された Container Insights でサポートされている詳細なメトリクスは収集できません。

Container Insights の設定方法は、クラスターが Amazon EC2 インスタンスでホストされているか、または AWS Fargate (Fargate) でホストされているかによって異なります。

Amazon EC2 でホストされている Amazon EKS クラスター

まだ行っていない場合は、必要な IAM ロールを含む前提条件を満たしていることを確認してください。詳細については、「前提条件を確認する」を参照してください。

Amazon は、Amazon EC2 で Amazon Elastic Kubernetes Service のモニターリングを設定するために使用できる Helm チャートを提供します。このモニターリングでは、メトリクスに AWS Distro for OpenTelemetry(ADOT) Collector を使用し、ログに Fluent Bit を使用します。したがって、Helm チャートは Amazon EC2 で Amazon EKS を使用し、CloudWatch Container Insights に送信するメトリクスとログを収集したいお客様にとって有用です。この Helm チャートの詳細については、「ADOT Helm chart for EKS on EC2 metrics and logs to Amazon CloudWatch Container Insights」(EKS on EC2 メトリクスと Amazon CloudWatch Container Insights へのログに関する ADOT Helm チャート) を参照してください。

または、このセクションの残りの手順を使用することもできます。

まず、次のコマンドを入力して、AWS Distro for OpenTelemetry コレクターを DaemonSet としてデプロイします。

```
curl https://raw.githubusercontent.com/aws-observability/aws-otel-collector/main/
deployment-template/eks/otel-container-insights-infra.yaml |
kubectl apply -f -
```

コレクターが実行中であることを確認するには、次のコマンドを使用します。

```
kubectl get pods -l name=aws-otel-eks-ci -n aws-otel-eks
```

このコマンドの出力に Running 状態の複数のポッドが含まれている場合、コレクターは実行中であり、クラスターからメトリクスを収集しています。コレクターは aws/containerinsights/*cluster-name*/performance という名前のロググループを作成し、パフォーマンスログイベントを送信します。

CloudWatch で Container Insights メトリクスを表示する方法については、Container Insights メトリクスの表示 を参照してください。

AWS は、このシナリオの GitHub に関するドキュメントも提供しています。Container Insights によって公開されるメトリクスとログをカスタマイズする場合は、https://aws-otel.github.io/docs/getting-started/container-insights/eks-infra を参照してください。

Fargate でホストされている Amazon EKS クラスター

ADOT Collector を設定およびデプロイして、Fargate 上の Amazon EKS クラスターにデプロイされたワークロードからシステムメトリクスを収集し、CloudWatch Container Insights に送信する方法については、AWS Distro for OpenTelemetry ドキュメントの「Container Insights EKS Fargate」を参照してください。

CloudWatch Logs にログを送信する

コンテナから Amazon CloudWatch Logs にログを送信するには、Fluent Bit または Fluentd を使用します。詳細については、「Fluent Bit」および「Fluentd」をご参照ください。

Fluentd をまだ使用していない場合は、次の理由により Fluent Bit を使用することをお勧めします。

- Fluent Bit はリソースフットプリントが小さく、Fluentd よりもメモリと CPU 使用率でリソース効率が高くなります。より詳細な比較については、「Fluent Bit と Fluentd のパフォーマンスの比較」を参照してください。
- Fluent Bit イメージは、 によって開発され、維持されていますAWS これにより、AWS が新しい Fluent Bit イメージ機能を採用し、問題に迅速に対応することが可能になります。

トピック

- Fluent Bit と Fluentd のパフォーマンスの比較
- CloudWatch Logs へログを送信する DaemonSet として Fluent Bit を設定する
- (オプション) CloudWatch Logs へログを送信する DaemonSet として Fluentd を設定する
- (オプション) Amazon EKS コントロールプレーンのログ記録の設定
- (オプション) App Mesh Envoy アクセスログを有効にする
- (オプション) 大規模なクラスターで Use_Kubelet 機能を有効にします。

Fluent Bit と Fluentd のパフォーマンスの比較

次の表は、メモリと CPU の使用率において、Fluent Bit が Fluentd よりも優れたパフォーマンスを発揮することを示しています。以下の数字は参考のためであり、環境によって異なる場合があります。

ログ/秒	Fluentd の CPU 使用率	Fluentd 互換設定での Fluent Bit の CPU 使用率	最適化された設定での Fluent Bit の CPU 使用率
100	0.35 vCPU	0.02 vCPU	0.02 vCPU
1,000	0.32 vCPU	0.14 vCPU	0.11 vCPU
5,000	0.85 vCPU	0.48 vCPU	0.30 vCPU
10,000	0.94 vCPU	0.60 vCPU	0.39 vCPU

ログ/秒	Fluentd のメモリ使用量	Fluentd 互換設定での Fluent Bit のメモリ使用量	最適化された設定での Fluent Bit のメモリ使用量
100	153 MB	46 MB	37 MB
1,000	270 MB	45 MB	40 MB
5,000	320 MB	55 MB	45 MB
10,000	375 MB	92 MB	75 MB

CloudWatch Logs へログを送信する DaemonSet として Fluent Bit を設定する

以下のセクションでは、Fluent Bit をデプロイして、コンテナから CloudWatch Logs にログを送信する方法について説明します。

トピック

Fluentd を既に使用している場合の違い

既に Fluentd を使用してコンテナから CloudWatch Logs にログを送信している場合は、このセクションを読んで Fluentd と Fluent Bit の違いを確認してください。Container Insights で Fluentd をまだ使用していない場合は、Fluent Bit の設定 に進むことができます。

Fluent Bit では、次の 2 つのデフォルト設定を提供しています。

- Fluent Bit 最適化設定 – Fluent Bit のベストプラクティスと整合する設定。
- Fluentd 互換設定 – 可能な限り Fluentd の動作に整合的なものとされた設定。

以下のリストでは、Fluentd と各 Fluent Bit の設定の違いが詳細に示されています。

- ログストリーム名の違い – Fluent Bit 最適化設定を使用する場合、ログストリーム名は異なります。

  [/aws/containerinsights/Cluster_Name/application]
  - Fluent Bit 最適化設定は、ログを *kubernetes-nodeName*-application.var.log.containers.*kubernetes-podName*_*kubernetes-namespace*_*kubernetes-container-name*-*kubernetes-containerID* に送信します
  - Fluentd は、ログを *kubernetes-podName*_*kubernetes-namespace*_*kubernetes-containerName*_*kubernetes-containerID* に送信します

  [/aws/containerinsights/Cluster_Name/host]
  - Fluent Bit 最適化設定は、ログを *kubernetes-nodeName*.*host-log-file* に送信します
  - Fluentd は、ログを *host-log-file*-*Kubernetes-NodePrivateIp* に送信します

  [/aws/containerinsights/Cluster_Name/dataplane]
  - Fluent Bit 最適化設定は、ログを *kubernetes-nodeName*.*dataplaneServiceLog* に送信します
  - Fluentd は、ログを *dataplaneServiceLog*-*Kubernetes-nodeName* に送信します
- Container Insights によって書き込まれる kube-proxy および aws-node のログファイルは、異なる場所にあります。Fluentd 設定の場合、これらのファイルは /aws/containerinsights/Cluster_Name/application に置かれます。Fluent Bit 最適化設定では、これらのファイルは /aws/containerinsights/Cluster_Name/dataplane に置かれます。
- pod_name や namespace_name などのメタデータのほとんどは Fluent Bit と Fluentd で同じですが、以下の点が異なります。

- Fluent Bit 最適化設定では docker_id を使用し、Fluentd では Docker.container_id を使用します。
- いずれの Fluent Bit 設定も、次のメタデータを使用しません。これらは Fluentd にのみ存在します: container_image_id、master_url、namespace_id、および namespace_labels。

Fluent Bit の設定

コンテナからログを収集するように Fluent Bit を設定するには、「Amazon EKS および Kubernetes の Container Insights のクイックスタートセットアップ」のステップを実行するか、このセクションのステップを実行します。

どちらの方法でも、クラスターノードにアタッチされた IAM ロールに十分なアクセス許可が必要です。Amazon EKS クラスターの実行に必要なアクセス許可の詳細については、Amazon EKS ユーザーガイドの「Amazon EKS IAM ポリシー、ロール、アクセス許可」を参照してください。

以下のステップでは、CloudWatch Logs へログを送信する daemonSet として Fluent Bit を設定します。このステップを完了すると、Fluent Bit は、次のロググループを作成します (まだ存在していない場合)。

> **Important**
>
> Container Insights で既に FluentD を設定しており、FluentD DaemonSet が期待どおりに実行されない場合 (containerd ランタイムを使用していると発生する場合があります)、Fluent Bit をインストールする前にアンインストールして、Fluent Bit が FluentD エラーログメッセージを処理しないようにする必要があります。それ以外の場合は、Fluent Bit が正常にインストールされたらすぐに FluentD をアンインストールする必要があります。Fluent Bit のインストール後に Fluentd をアンインストールすることで、この移行プロセス中のログの継続を確保できます。ログを CloudWatch Logs に送信するには、Fluent Bit または FluentD のいずれか 1 つのみが必要になります。

ロググループ名	ログソース
/aws/containerinsights/ *Cluster_Name* /application	/var/log/containers のすべてのログファイル

ロググループ名	ログソース
/aws/containerinsights/ *Cluster_Name* /host	/var/log/dmesg 、/var/log/secure 、および /var/log/messages からのログ
/aws/containerinsights/ *Cluster_Name* /dataplane	/var/log/journal 、kubelet.service 、およびkubeproxy.service に対する docker.service のログ。

Fluent Bit をインストールしてコンテナから CloudWatch Logs にログを送信するには

1. amazon-cloudwatch という名前の名前空間がまだない場合は、次のコマンドを入力して作成します。

```
kubectl apply -f https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-
container-insights/latest/k8s-deployment-manifest-templates/deployment-mode/
daemonset/container-insights-monitoring/cloudwatch-namespace.yaml
```

2. 次のコマンドを実行して、クラスター名とログを送信するリージョンを持つ cluster-info という名前の ConfigMap を作成します。*cluster-name* と *cluster-region* をクラスターの名前とリージョンに置き換えます。

```
ClusterName=cluster-name
RegionName=cluster-region
FluentBitHttpPort='2020'
FluentBitReadFromHead='Off'
[[ ${FluentBitReadFromHead} = 'On' ]] && FluentBitReadFromTail='Off'||
 FluentBitReadFromTail='On'
[[ -z ${FluentBitHttpPort} ]] && FluentBitHttpServer='Off' ||
 FluentBitHttpServer='On'
kubectl create configmap fluent-bit-cluster-info \
--from-literal=cluster.name=${ClusterName} \
--from-literal=http.server=${FluentBitHttpServer} \
--from-literal=http.port=${FluentBitHttpPort} \
--from-literal=read.head=${FluentBitReadFromHead} \
--from-literal=read.tail=${FluentBitReadFromTail} \
--from-literal=logs.region=${RegionName} -n amazon-cloudwatch
```

このコマンドでは、プラグインメトリクスをモニターリングするための FluentBitHttpServer がデフォルトでオンになっています。無効にするには、コマンドの 3 行目を FluentBitHttpPort='' (空の文字列) に変更します。

また、デフォルトでは、Fluent Bit はテールからログファイルを読み取り、デプロイ後に新しいログのみを取得します。逆をご希望の場合は、FluentBitReadFromHead='On' を設定することで、ファイルシステム内のすべてのログが収集されます。

3. 次のいずれかのコマンドを実行して、Fluent Bit daemonset をクラスターにダウンロードしてデプロイします。

   - Linux コンピュータ用の Fluent Bit 最適化設定が必要な場合は、このコマンドを実行します。

     ```
     kubectl apply -f https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-insights/latest/k8s-deployment-manifest-templates/deployment-mode/daemonset/container-insights-monitoring/fluent-bit/fluent-bit.yaml
     ```

   - Windows コンピュータ用の Fluent Bit 最適化設定が必要な場合は、このコマンドを実行します。

     ```
     kubectl apply -f https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-insights/latest/k8s-deployment-manifest-templates/deployment-mode/daemonset/container-insights-monitoring/fluent-bit/fluent-bit-windows.yaml
     ```

   - Linux コンピュータを使用しており、より Fluentd に似ている Fluent Bit の設定が必要な場合は、このコマンドを実行します。

     ```
     kubectl apply -f https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-insights/latest/k8s-deployment-manifest-templates/deployment-mode/daemonset/container-insights-monitoring/fluent-bit/fluent-bit-compatible.yaml
     ```

> **Important**
>
> Fluent Bit デーモンセットの設定では、デフォルトでログレベルが INFO に設定されるため、CloudWatch Logs の取り込みコストが高くなる可能性があります。ログの取り込み量とコストを削減したい場合は、ログレベルを ERROR に変更すると削減できます。

ログの量を減らす方法の詳細については、「(オプション) Fluent Bit からのログボリュームの縮小」を参照してください

4. 次のコマンドを実行してデプロイを検証します。各ノードには、fluent-bit-* という名前の 1 つのポッドが必要です。

```
kubectl get pods -n amazon-cloudwatch
```

上記の手順を実行することにより、クラスターに次のリソースが作成されます。

- `Fluent-Bit` 名前空間の `amazon-cloudwatch` という名前のサービスアカウント。このサービスアカウントは、Fluent Bit daemonSet を実行するために使用されます。詳細については、Kubernetes Reference のサービスアカウントの管理を参照してください。
- `Fluent-Bit-role` 名前空間の `amazon-cloudwatch` という名前のクラスターロール。このクラスターロールは、ポッドログの `get`、`list`、`watch` の各アクセス許可を `Fluent-Bit` サービスアカウントに付与します。詳細については、Kubernetes Reference の API の概要を参照してください。
- `Fluent-Bit-config` 名前空間の `amazon-cloudwatch` という名前の ConfigMap。この ConfigMap には、Fluent Bit によって使用される設定が含まれています。詳細については、Kubernetes Tasks ドキュメントの「Configure a Pod to Use a ConfigMap」を参照してください。

Fluent Bit の設定を検証する場合は、次の手順を実行します。

Fluent Bit 設定の検証

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで、[Log groups] (ロググループ) を選択します。
3. Fluent Bit をデプロイしたリージョンにいることを確認してください。
4. リージョン内のロググループのリストを確認します。次のように表示されます。
   - `/aws/containerinsights/Cluster_Name/application`
   - `/aws/containerinsights/Cluster_Name/host`
   - `/aws/containerinsights/Cluster_Name/dataplane`

5. これらのロググループのいずれかに移動し、ログストリームの [Last Event Time] (最終のイベント時刻) を確認します。Fluent Bit をデプロイしたときと比べて最近のものである場合は、設定が検証されます。

   /dataplane ロググループの作成にわずかな遅延が生じることがあります。Fluent Bit がそのロググループのログ送信を開始したときにのみ、これらのロググループが作成されるため、これは正常です。

複数行ログのサポート

複数行ログで Fluent Bit を使用する方法については、Fluent Bit のドキュメントの次のセクションを参照してください。

- 複数行の解析
- 複数行とコンテナ (v1.8)
- 複数行コア (v1.8)
- tail 入力で常に複数行を使用する

(オプション) Fluent Bit からのログボリュームの縮小

デフォルトでは、Fluent Bit アプリケーションログおよび Kubernetes メタデータを CloudWatch に送信します。CloudWatch に送信されるデータの量を減らす場合は、これらのデータソースが一方または両方の CloudWatch に送信されることを停止できます。

Fluent Bit アプリケーションログを停止するには、Fluent-Bit.yaml ファイルから次のセクションを削除します。

```
[INPUT]
        Name                tail
        Tag                 application.*
        Path                /var/log/containers/fluent-bit*
        Parser              docker
        DB                  /fluent-bit/state/flb_log.db
        Mem_Buf_Limit       5MB
        Skip_Long_Lines     On
        Refresh_Interval    10
```

CloudWatch に送信されたログイベントに Kubernetes メタデータが追加されないように削除するには、application-log.conf ファイルの Fluent-Bit.yaml セクションに以下のフィルターを追加します。*<Metadata_1>* および同様のフィールドを実際のメタデータ識別子に置き換えます。

```
application-log.conf: |
    [FILTER]
        Name                nest
        Match               application.*
        Operation           lift
        Nested_under        kubernetes
        Add_prefix          Kube.

    [FILTER]
        Name                modify
        Match               application.*
        Remove              Kube.<Metadata_1>
        Remove              Kube.<Metadata_2>
        Remove              Kube.<Metadata_3>

    [FILTER]
        Name                nest
        Match               application.*
        Operation           nest
        Wildcard            Kube.*
        Nested_under        kubernetes
        Remove_prefix       Kube.
```

トラブルシューティング

正しいリージョンで確認しているが、これらのロググループが表示されない場合は、Fluent Bit daemonSet ポッドのログでエラーを確認します。

次のコマンドを実行してステータスが Running であることを確認します。

```
kubectl get pods -n amazon-cloudwatch
```

IAM アクセス許可に関連するエラーがログにある場合は、クラスターノードにアタッチされた IAM ロールを確認します。Amazon EKS クラスターの実行に必要なアクセス許可の詳細については、Amazon EKS ユーザーガイドの「Amazon EKS IAM ポリシー、ロール、アクセス許可」を参照してください。

ポッドのステータスが CreateContainerConfigError である場合は、次のコマンドを実行して正確なエラーを取得します。

```
kubectl describe pod pod_name -n amazon-cloudwatch
```

ダッシュボード

実行中の各プラグインのメトリクスをモニターリングするダッシュボードを作成できます。入力バイトと出力バイト、レコード処理レート、出力エラー、およびリトライ/失敗率のデータを表示できます。これらのメトリクスを表示するには、Amazon EKS と Kubernetes クラスターの Prometheus メトリクスコレクションを使用して CloudWatch エージェントをインストールする必要があります。ダッシュボードの設定方法の詳細については、「Amazon EKS および Kubernetes クラスターに Prometheus メトリクスコレクションを使用して CloudWatch エージェントをインストールする」を参照してください。

> **Note**
>
> このダッシュボードを設定する前に、Prometheus メトリクスの Container Insights を設定する必要があります。詳細については、「Container Insights の Prometheus メトリクスのモニターリング」を参照してください。

Fluent Bit Prometheus メトリクスのダッシュボードを作成するには

1. 環境変数を作成し、次の行の右側の値をデプロイと一致するように置き換えます。

```
DASHBOARD_NAME=your_cw_dashboard_name
REGION_NAME=your_metric_region_such_as_us-west-1
CLUSTER_NAME=your_kubernetes_cluster_name
```

2. 次のコマンドを実行して、ダッシュボードを作成します。

```
curl https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-
insights/latest/k8s-deployment-manifest-templates/deployment-mode/service/cwagent-
prometheus/sample_cloudwatch_dashboards/fluent-bit/cw_dashboard_fluent_bit.json \
| sed "s/{{YOUR_AWS_REGION}}/${REGION_NAME}/g" \
| sed "s/{{YOUR_CLUSTER_NAME}}/${CLUSTER_NAME}/g" \
| xargs -0 aws cloudwatch put-dashboard --dashboard-name ${DASHBOARD_NAME} --
dashboard-body
```

(オプション) CloudWatch Logs へログを送信する DaemonSet として Fluentd を設定する

> **Warning**
>
> Container Insights の Fluentd のサポートは現在メンテナンスモードになっています。つまり、AWS はこれ以上 Fluentd の更新を提供せず、近い将来に廃止する予定です。さらに、Container Insights の現在の Fluentd 構成は、最新の改善およびセキュリティパッチがない古いバージョンの Fluentd Image `fluent/fluentd-kubernetes-daemonset:v1.10.3-debian-cloudwatch-1.0` を使用しています。オープンソースコミュニティでサポートされている最新の Fluentd イメージについては、「fluentd-kubernetes-daemonset」を参照してください。
>
> 可能な限り Container Insights で FluentBit を使用するように移行することを強くお勧めします。Container Insights のログフォワーダとして FluentBit を使用すると、パフォーマンスが大幅に向上します。
>
> 詳細については、「CloudWatch Logs へログを送信する DaemonSet として Fluent Bit を設定する」および「 Fluentd を既に使用している場合の違い」を参照してください。

コンテナからログを収集するように Fluentd をセットアップするには、「Amazon EKS および Kubernetes の Container Insights のクイックスタートセットアップ」のステップに従うか、このセクションのステップに従います。以下のステップでは、CloudWatch Logs へログを送信する DaemonSet として Fluentd を設定します。このステップを完了すると、Fluentd は、まだ存在していない場合は次のロググループを作成します。

ロググループ名	ログソース
`/aws/containerinsights/`*`Cluster_Name`*`/application`	`/var/log/containers` のすべてのログファイル
`/aws/containerinsights/`*`Cluster_Name`*`/host`	`/var/log/dmesg` 、`/var/log/secure` 、および `/var/log/messages` からのログ
`/aws/containerinsights/`*`Cluster_Name`*`/dataplane`	`/var/log/journal` 、`kubelet.service` 、および`kubeproxy.service` に対する `docker.service` のログ。

ステップ 1: CloudWatch の名前空間を作成する

CloudWatch に対して amazon-cloudwatch という Kubernetes 名前空間を作成するには、次のステップを使用します。すでにこの名前空間を作成している場合は、以下のステップをスキップできます。

CloudWatch の名前空間を作成するには

- 以下のコマンドを入力します。

```
kubectl apply -f https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-insights/latest/k8s-deployment-manifest-templates/deployment-mode/daemonset/container-insights-monitoring/cloudwatch-namespace.yaml
```

ステップ 2: Fluentd をインストールする

Fluentd をダウンロードして、このプロセスを開始します。これらのステップを完了すると、クラスターで次のリソースが作成されます。

- fluentd 名前空間の amazon-cloudwatch という名前のサービスアカウント。このサービスアカウントは、DaemonSet Fluentd を実行するために使用されます。詳細については、Kubernetes Reference のサービスアカウントの管理を参照してください。
- fluentd 名前空間の amazon-cloudwatch という名前のクラスターロール。このクラスターロールは、ポッドログの get、list、watch の各アクセス許可を fluentd サービスアカウントに付与します。詳細については、Kubernetes Reference の API の概要を参照してください。
- fluentd-config 名前空間の amazon-cloudwatch という名前の ConfigMap。この ConfigMap には、Fluentd によって使用される設定が含まれています。詳細については、Kubernetes Tasks ドキュメントの「Configure a Pod to Use a ConfigMap」を参照してください。

Fluentd をインストールするには

1. クラスター名を持つ cluster-info という名前の ConfigMap と、ログの送信先となる AWS リージョンを作成します。次のコマンドを実行し、プレースホルダーをクラスター名とリージョン名で更新します。

```
kubectl create configmap cluster-info \
--from-literal=cluster.name=cluster_name \
--from-literal=logs.region=region_name -n amazon-cloudwatch
```

2. 次のコマンドを実行して、Fluentd DaemonSet をクラスターにダウンロードしてデプロイします。正しいアーキテクチャでコンテナイメージを使用していることを確認してください。サンプルマニフェストは x86 インスタンスでのみ機能し、クラスターに Advanced RISC Machine (ARM) インスタンスがある場合に `CrashLoopBackOff` を入力します。Fluentd daemonSet には、複数の基礎となるイメージに 1 つのタグを使用し、コンテナランタイムが適切なイメージをプルすることを可能にする、公式のマルチアーキテクチャ Docker イメージはありません。Fluentd ARM イメージでは、`arm64` サフィックスが付いた別のタグが使用されます。

```
kubectl apply -f https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-
container-insights/latest/k8s-deployment-manifest-templates/deployment-mode/
daemonset/container-insights-monitoring/fluentd/fluentd.yaml
```

> **Note**
>
> Fluentd 構成を最適化し、Kubernetes API エンドポイントに対する Fluentd API リクエストの影響を最小限に抑えるための最近の変更により、Kubernetes フィルターの「監視」オプションはデフォルトで無効になっています。詳細については、fluent-plugin-kubernetes_metadata_filter を参照してください。

3. 次のコマンドを実行してデプロイを検証します。各ノードには、`fluentd-cloudwatch-*` という名前の 1 つのノードが必要です。

```
kubectl get pods -n amazon-cloudwatch
```

ステップ 3: Fluentd のセットアップを検証する

Fluentd のセットアップを検証するには、次のステップを使用します。

Container Insights の Fluentd セットアップを検証するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで、[Log groups] (ロググループ) を選択します。Fluentd をコンテナにデプロイしたリージョンを選択していることを確認してください。

   リージョンのロググループのリストは、以下のように表示されます。

   - `/aws/containerinsights/Cluster_Name/application`
   - `/aws/containerinsights/Cluster_Name/host`

- /aws/containerinsights/*Cluster_Name*/dataplane

これらのロググループが表示される場合、Fluentd のセットアップは検証済みです。

複数行ログのサポート

2019 年 8 月 19 日に、Fluentd によって収集されたログの複数行のログサポートを追加しました。

デフォルトでは、複数行のログエントリスターターは、空白のない任意の文字です。つまり、空白文字のない文字で始まるすべてのログ行は、新しい複数行のログエントリとみなされます。

独自のアプリケーションログで異なる複数行のスターターを使用している場合は、fluentd.yaml ファイルで 2 つの変更を行うことでそれらのログをサポートできます。

まず、exclude_path の containers セクションの fluentd.yaml フィールドにログファイルのパス名を追加することで、デフォルトの複数行サポートから除外します。次に例を示します。

```
<source>
  @type tail
  @id in_tail_container_logs
  @label @containers
  path /var/log/containers/*.log
  exclude_path ["full_pathname_of_log_file*", "full_pathname_of_log_file2*"]
```

次に、ログファイルのブロックを fluentd.yaml ファイルに追加します。以下の例は、複数行のスターターとしてタイムスタンプの正規表現を使用する CloudWatch エージェントのログファイルに使用されます。このブロックをコピーして fluentd.yaml に追加できます。示された行を変更して、使用するアプリケーションログファイル名と複数行のスターターを反映します。

```
<source>
  @type tail
  @id in_tail_cwagent_logs
  @label @cwagentlogs
  path /var/log/containers/cloudwatch-agent*
  pos_file /var/log/cloudwatch-agent.log.pos
  tag *
  read_from_head true
 <parse>
    @type json
```

```
    time_format %Y-%m-%dT%H:%M:%S.%NZ
 </parse>
</source>
```

```
<label @cwagentlogs>
  <filter **>
    @type kubernetes_metadata
    @id filter_kube_metadata_cwagent
  </filter>

  <filter **>
    @type record_transformer
    @id filter_cwagent_stream_transformer
    <record>
      stream_name ${tag_parts[3]}
    </record>
 </filter>

  <filter **>
    @type concat
    key log
    multiline_start_regexp /^\d{4}[-/]\d{1,2}[-/]\d{1,2}/
    separator ""
    flush_interval 5
    timeout_label @NORMAL
 </filter>

 <match **>
    @type relabel
    @label @NORMAL
 </match>
</label>
```

(オプション) Fluentd からのログボリュームの縮小

デフォルトでは、Fluentd アプリケーションログおよび Kubernetes メタデータを CloudWatch に送信します。CloudWatch に送信されるデータの量を減らす場合は、これらのデータソースが一方または両方の CloudWatch に送信されることを停止できます。

Fluentd アプリケーションログを停止するには、`fluentd.yaml` ファイルから次のセクションを削除します。

```
<source>
  @type tail
  @id in_tail_fluentd_logs
  @label @fluentdlogs
  path /var/log/containers/fluentd*
  pos_file /var/log/fluentd.log.pos
  tag *
  read_from_head true
   <parse>
   @type json
   time_format %Y-%m-%dT%H:%M:%S.%NZ
  </parse>
</source>

<label @fluentdlogs>
  <filter **>
    @type kubernetes_metadata
    @id filter_kube_metadata_fluentd
  </filter>

  <filter **>
    @type record_transformer
    @id filter_fluentd_stream_transformer
    <record>
      stream_name ${tag_parts[3]}
    </record>
   </filter>

   <match **>
     @type relabel
     @label @NORMAL
   </match>
</label>
```

CloudWatch に送信されたログイベントに Kubernetes メタデータが追加されないように削除するには、 `record_transformer` ファイルの `fluentd.yaml` セクションに 1 行を追加します。このメタデータを削除するログソースに、次の行を追加します。

```
remove_keys $.kubernetes.pod_id, $.kubernetes.master_url,
 $.kubernetes.container_image_id, $.kubernetes.namespace_id
```

例:

```
<filter **>
  @type record_transformer
  @id filter_containers_stream_transformer
  <record>
    stream_name ${tag_parts[3]}
  </record>
  remove_keys $.kubernetes.pod_id, $.kubernetes.master_url,
 $.kubernetes.container_image_id, $.kubernetes.namespace_id
</filter>
```

トラブルシューティング

正しいリージョンで確認しているものの、これらのロググループが表示されない場合は、Fluentd DaemonSet ポッドのログでエラーを確認します。

次のコマンドを実行してステータスが Running であることを確認します。

```
kubectl get pods -n amazon-cloudwatch
```

前のコマンドの結果で、ポッドの名前が fluentd-cloudwatch で始まることに注意してください。次のコマンドでこのポッド名を使用します。

```
kubectl logs pod_name -n amazon-cloudwatch
```

ログに IAM アクセス権限に関連するエラーがある場合は、クラスターノードにアタッチされた IAM ロールを確認します。Amazon EKS クラスターの実行に必要なアクセス許可の詳細については、Amazon EKS ユーザーガイドの「Amazon EKS IAM ポリシー、ロール、アクセス許可」を参照してください。

ポッドのステータスが CreateContainerConfigError である場合は、次のコマンドを実行して正確なエラーを取得します。

```
kubectl describe pod pod_name -n amazon-cloudwatch
```

pod の状態が CrashLoopBackOff の場合は、Fluentd コンテナイメージのアーキテクチャが Fluentd をインストールしたときのノードと同じであることを確認します。クラスターに x86 ノー

ドと ARM64 ノードの両方がある場合は、kubernetes.io/arch ラベルを使用して、イメージを正しいノードに配置できます。詳細については、kuberntes.io/arch をご参照ください。

(オプション) Amazon EKS コントロールプレーンのログ記録の設定

Amazon EKS を使用している場合は、オプションで Amazon EKS コントロールプレーンのログ記録を有効にして、Amazon EKS コントロールプレーンから CloudWatch Logs に監査ログと診断ログを直接提供することができます。詳細については、「Amazon EKS コントロールプレーンのログ記録」を参照してください。

(オプション) App Mesh Envoy アクセスログを有効にする

App Mesh Envoy アクセスログを CloudWatch Logs に送信するように Container Insights Fluentd を設定できます。詳細については、「Logging」(ログ記録) を参照してください。

Envoy アクセスログを CloudWatch Logs に送信するには

1. クラスターで Fluentd を設定します。詳細については、「(オプション) CloudWatch Logs へログを送信する DaemonSet として Fluentd を設定する」を参照してください。
2. 仮想ノードの Envoy アクセスログを設定します。手順については、「Logging」(ログ記録) を参照してください。必ず、ログパスを各仮想ノードで **/dev/stdout** に設定してください。

完了すると、Envoy アクセスログが /aws/containerinsights/*Cluster_Name*/application ロググループに送信されます。

(オプション) 大規模なクラスターで Use_Kubelet 機能を有効にします。

デフォルトでは、FluentBit Kubernetes プラグインでは Use_Kubelet 機能は無効になっています。この機能を有効にすると、API サーバーへのトラフィックが減少し、API サーバーがボトルネックになる問題を軽減できます。大規模なクラスターに対してこの機能を有効にすることをお勧めします。

Use_Kubelet を有効にするには、まずノードとノード/プロキシのアクセス許可を clusterRole 設定に追加します。

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: fluent-bit-role
rules:
  - nonResourceURLs:
      - /metrics
```

```
    verbs:
      - get
  - apiGroups: [""]
    resources:
      - namespaces
      - pods
      - pods/logs
      - nodes
      - nodes/proxy
    verbs: ["get", "list", "watch"]
```

DaemonSet 設定では、この機能にはホストネットワークアクセスが必要です。amazon/aws-for-fluent-bit のイメージバージョンが 2.12.0 以降か、Fluent Bit のイメージバージョンが 1.7.2 以降である必要があります。

```
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: fluent-bit
  namespace: amazon-cloudwatch
  labels:
    k8s-app: fluent-bit
    version: v1
    kubernetes.io/cluster-service: "true"
spec:
  selector:
    matchLabels:
      k8s-app: fluent-bit
  template:
    metadata:
      labels:
        k8s-app: fluent-bit
        version: v1
        kubernetes.io/cluster-service: "true"
    spec:
      containers:
      - name: fluent-bit
        image: amazon/aws-for-fluent-bit:2.19.0
        imagePullPolicy: Always
        env:
            - name: AWS_REGION
              valueFrom:
                configMapKeyRef:
```

```
                name: fluent-bit-cluster-info
                key: logs.region
          - name: CLUSTER_NAME
            valueFrom:
              configMapKeyRef:
                name: fluent-bit-cluster-info
                key: cluster.name
          - name: HTTP_SERVER
            valueFrom:
              configMapKeyRef:
                name: fluent-bit-cluster-info
                key: http.server
          - name: HTTP_PORT
            valueFrom:
              configMapKeyRef:
                name: fluent-bit-cluster-info
                key: http.port
          - name: READ_FROM_HEAD
            valueFrom:
              configMapKeyRef:
                name: fluent-bit-cluster-info
                key: read.head
          - name: READ_FROM_TAIL
            valueFrom:
              configMapKeyRef:
                name: fluent-bit-cluster-info
                key: read.tail
          - name: HOST_NAME
            valueFrom:
              fieldRef:
                fieldPath: spec.nodeName
          - name: HOSTNAME
            valueFrom:
              fieldRef:
                apiVersion: v1
                fieldPath: metadata.name
          - name: CI_VERSION
            value: "k8s/1.3.8"
        resources:
          limits:
            memory: 200Mi
          requests:
            cpu: 500m
            memory: 100Mi
```

```
        volumeMounts:
        # Please don't change below read-only permissions
        - name: fluentbitstate
          mountPath: /var/fluent-bit/state
        - name: varlog
          mountPath: /var/log
          readOnly: true
        - name: varlibdockercontainers
          mountPath: /var/lib/docker/containers
          readOnly: true
        - name: fluent-bit-config
          mountPath: /fluent-bit/etc/
        - name: runlogjournal
          mountPath: /run/log/journal
          readOnly: true
        - name: dmesg
          mountPath: /var/log/dmesg
          readOnly: true
      terminationGracePeriodSeconds: 10
      hostNetwork: true
      dnsPolicy: ClusterFirstWithHostNet
      volumes:
      - name: fluentbitstate
        hostPath:
          path: /var/fluent-bit/state
      - name: varlog
        hostPath:
          path: /var/log
      - name: varlibdockercontainers
        hostPath:
          path: /var/lib/docker/containers
      - name: fluent-bit-config
        configMap:
          name: fluent-bit-config
      - name: runlogjournal
        hostPath:
          path: /run/log/journal
      - name: dmesg
        hostPath:
          path: /var/log/dmesg
      serviceAccountName: fluent-bit
      tolerations:
      - key: node-role.kubernetes.io/master
        operator: Exists
```

```
        effect: NoSchedule
      - operator: "Exists"
        effect: "NoExecute"
      - operator: "Exists"
        effect: "NoSchedule"
```

Kubernetes プラグインの設定は次のようになります。

```
[FILTER]
        Name                kubernetes
        Match               application.*
        Kube_URL            https://kubernetes.default.svc:443
        Kube_Tag_Prefix     application.var.log.containers.
        Merge_Log           On
        Merge_Log_Key       log_processed
        K8S-Logging.Parser  On
        K8S-Logging.Exclude Off
        Labels              Off
        Annotations         Off
        Use_Kubelet         On
        Kubelet_Port        10250
        Buffer_Size         0
```

Amazon EKS および Kubernetes での Container Insights の更新または削除

これらのセクションのステップを使用して、CloudWatch エージェントコンテナイメージを更新するか、Container Insights を Amazon EKS または Kubernetes クラスターから削除します。

トピック

- Amazon EKS 向けにオブザーバビリティが強化された Container Insights へのアップグレード
- CloudWatch エージェントコンテナイメージの更新
- Container Insights の CloudWatch エージェントと Fluent Bit の削除

Amazon EKS 向けにオブザーバビリティが強化された Container Insights へのアップグレード

> **Important**
>
> Amazon EKS クラスターに Container Insights をアップグレードまたはインストールする場合は、このセクションの手順を使用するのではなく Amazon CloudWatch Observability EKS アドオンを使用してインストールすることをお勧めします。また、高速コンピューティング

メトリクスを取得するには、Amazon CloudWatch Observability EKS アドオンを使用する必要があります。詳細と手順については、「Amazon CloudWatch Observability EKS アドオンのインストール」を参照してください。

Amazon EKS 向けにオブザーバビリティが強化された Container Insights は、Container Insights の最新バージョンです。Amazon EKS を実行しているクラスターから詳細なメトリクスが収集され、キュレーションされたすぐに使用できるダッシュボードが提供されます。これにより、アプリケーションやインフラストラクチャテレメトリを掘り下げて調べることができます。Container Insights のこのバージョンについての詳細は、「Amazon EKS 向けにオブザーバビリティが強化された Container Insights」を参照してください。

Amazon EKS クラスターに Container Insights の元のバージョンをインストールしており、オブザーバビリティが強化された新しいバージョンにアップグレードしたい場合は、このセクションの手順に従ってください。

Important

このセクションに記載されている手順を完了する前に、cert-manager を含む前提条件を確認しておく必要があります。詳細については、「CloudWatch エージェントオペレーターと Fluent Bit を使用したクイックスタート」を参照してください。

Amazon EKS 向けにオブザーバビリティが強化された Container Insights にAmazon EKS クラスターをアップグレードするには

1. 次のコマンドを入力して、CloudWatch エージェントオペレーターをインストールします。*my-cluster-name* は Amazon EKS または Kubernetes クラスターの名前に置き換え、*my-cluster-region* はログが発行されるリージョンの名前に置き換えます。AWS アウトバウンドデータ転送コストを削減するために、クラスターがデプロイされているのと同じリージョンを使用することをお勧めします。

```
ClusterName=my-cluster-name
RegionName=my-cluster-region
curl https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-
container-insights/main/k8s-quickstart/cwagent-operator-rendered.yaml | sed 's/
{{cluster_name}}/'${ClusterName}'/g;s/{{region_name}}/'${RegionName}'/g' | kubectl
 apply -f -
```

リソースの競合によって生じるエラーが見られる場合は、CloudWatch エージェントと Fluent Bit で ServiceAccount、ClusterRole、ClusterRoleBinding などの関連コンポーネントがクラスターに既にインストールされていることが原因である可能性があります。CloudWatch エージェントオペレーターによって CloudWatch エージェントとその関連コンポーネントがインストールされるときに内容の変更が検出されると、デフォルトではインストールまたは更新が失敗します。これによって、クラスター上にあるリソースの状態が上書きされることを防いでいます。クラスターに以前にインストールしていた Container Insights セットアップを含む既存の CloudWatch エージェントをすべて削除してから、CloudWatch エージェントオペレーターをインストールすることをお勧めします。

2. (オプション) 既存のカスタム Fluent Bit 設定を適用するには、Fluent Bit daemonset に関連付けられている configmap を更新する必要があります。CloudWatch エージェントオペレーターは Fluent Bit 用のデフォルト設定を提供し、デフォルト設定は必要に応じて上書きまたは変更できます。カスタム設定を適用するには、以下の手順に従います。

   a. 次のコマンドを入力して、既存の設定を開きます。

   ```
   kubectl edit cm fluent-bit-config -n amazon-cloudwatch
   ```

   b. ファイルに変更を加え、`:wq` と入力してファイルを保存し、編集モードを終了します。

   c. 次のコマンドを入力して Fluent Bit を再起動します。

   ```
   kubectl rollout restart fluent-bit -n amazon-cloudwatch
   ```

CloudWatch エージェントコンテナイメージの更新

> **Important**
>
> Amazon EKS クラスターに Container Insights をアップグレードまたはインストールする場合は、このセクションの手順を使用するのではなく Amazon CloudWatch Observability EKS アドオンを使用してインストールすることをお勧めします。また、高速コンピューティングメトリクスを取得するには、Amazon CloudWatch Observability EKS アドオンまたは CloudWatch エージェントオペレーターを使用する必要があります。詳細と手順については、「Amazon CloudWatch Observability EKS アドオンのインストール」を参照してください。

コンテナイメージを最新バージョンに更新する必要がある場合は、このセクションの手順を使用します。

コンテナイメージを更新するには

1. 次のコマンドを入力して、amazoncloudwatchagent カスタマーリソース定義 (CRD) が既に存在しているかどうかを確認します。

   ```
   kubectl get crds amazoncloudwatchagents.cloudwatch.aws.amazon.com -n amazon-cloudwatch
   ```

   このコマンドで CRD がないというエラーが返される場合、クラスターには、CloudWatch エージェントオペレーターで設定された Amazon EKS 向けにオブザーバビリティが強化された Container Insights がありません。この場合は、Amazon EKS 向けにオブザーバビリティが強化された Container Insights へのアップグレード を参照してください。
2. 次のコマンドを入力して、最新の cwagent-version.yaml ファイルを適用します。

   ```
   curl https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-insights/main/k8s-quickstart/cwagent-version.yaml | kubectl apply -f -
   ```

Container Insights の CloudWatch エージェントと Fluent Bit の削除

Amazon EKS 向けの CloudWatch Observability アドオンのインストールを使用して Container Insights をインストールした場合、次のコマンドを入力して Container Insights と CloudWatch エージェントを削除できます。

> **Note**
>
> Amazon EKS アドオンは Windows ワーカーノードで Container Insights をサポートするようになりました。Amazon EKS アドオンを削除すると、Windows 用の Container Insights も削除されます。

```
aws eks delete-addon –cluster-name my-cluster –addon-name amazon-cloudwatch-observability
```

そうでない場合、CloudWatch エージェントおよび Fluent Bit に関連するすべてのリソースを削除するには、次のコマンドを入力します。このコマンドで、*My_Cluster_Name* はお使いの Amazon

EKS または Kubernetes クラスターの名前、*My_Region* はログが発行されるリージョンの名前です。

```
ClusterName=My_Cluster_Name
RegionName=My-Region
curl https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-
container-insights/main/k8s-quickstart/cwagent-operator-rendered.yaml | sed 's/
{{cluster_name}}/'${ClusterName}'/g;s/{{region_name}}/'${RegionName}'/g' | kubectl
 delete -f -
curl https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-
insights/main/k8s-quickstart/cwagent-custom-resource-definitions.yaml | kubectl delete
 -f -
```

Container Insights メトリクスの表示

Container Insights を設定してメトリクスを収集したら、CloudWatch コンソールでそれらのメトリクスを表示できます。

Container Insights のメトリクスがダッシュボードに表示されるようにするには、Container Insights のセットアップを完了する必要があります。詳細については、「Container Insights の設定」を参照してください。

この手順では、収集したログデータから Container Insights が自動的に生成するメトリクスを表示する方法について説明します。このセクションの残りの部分では、データについてさらに詳しく把握し、 CloudWatch Logs Insights を使用して、より詳細なレベルでより多くのメトリクスを表示する方法について説明します。

Container Insights のメトリクスを表示するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで、[インサイト]、[Contributor Insights] の順に選択します。
3. Container Insights の下のドロップダウンボックスで、[パフォーマンスモニタリング] を選択します。
4. 上部近くにあるドロップダウンボックスを使用して、表示するリソースのタイプと特定のリソースを選択します。

Container Insights が収集する任意のメトリクスに CloudWatch アラームを設定できます。詳細については、「 Amazon CloudWatch でのアラームの使用」を参照してください。

> **Note**
>
> コンテナ化されたアプリケーションをモニターリングするように CloudWatch Application Insights を既に設定している場合は、Container Insights ダッシュボードの下に Application Insights ダッシュボードが表示されます。Application Insights をまだ有効にしていない場合は、Container Insights ダッシュボードのパフォーマンスビューの下の [Auto-configure Application Insights] (Application Insights の自動設定) を選択して有効にできます。Application Insights とコンテナ化されたアプリケーションの詳細については、「Amazon ECS および Amazon EKS リソースのモニターリングで Application Insights を有効にする」を参照してください。

寄与度が上位の要素の表示

Container Insights によるパフォーマンスモニターリングの一部のビューでは、メモリまたは CPU、または最近アクティブにされたリソース別に、寄与度が高い要素を表示することもできます。これは、ページの上部近くにあるドロップダウンボックスで、次のいずれかのダッシュボードを選択すると表示されます。

- ECS サービス
- ECS タスク
- EKS 名前空間
- EKS サービス
- EKS ポッド

これらのリソースタイプから 1 つを表示すると、初期状態では CPU 使用率でソートされた表がページの下部に表示されます。この表は、メモリの使用量または最近のアクティビティでソートするように変更できます。表内の 1 つの行の詳細を表示するには、その行の横にあるチェックボックスをオンにし、[アクション] をクリックします。その後で、開いた [アクション] メニューからオプションの 1 つを選択します。

CloudWatch Logs Insights を使用して Container Insights データを表示する

Container Insights は、埋め込みメトリクス形式を使用して、パフォーマンスログイベントでメトリクスを収集します。ログは CloudWatch Logs に保存されます。CloudWatch は、CloudWatch コンソールで表示できるログから複数のメトリクスを自動的に生成します。また、CloudWatch Logs Insight クエリを使用して収集されるパフォーマンスデータをより深く分析することもできます。

CloudWatch Logs Insights の詳細については、「CloudWatch Logs Insights を使用したログデータの分析」を参照してください。クエリで使用できるログフィールドの詳細については、「Amazon EKS および Kubernetes の Container Insights パフォーマンスログイベント」を参照してください。

CloudWatch Logs Insights を使用してコンテナメトリクスデータをクエリするには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで、[Insights] を選択します。

   画面の上部にクエリエディタがあります。CloudWatch Logs Insights を初めて開くと、このボックスにデフォルトクエリが表示されます。デフォルトでは、最新の 20 件のログイベントが返されます。
3. クエリエディタの上部にあるボックスで、クエリの対象となるいずれかの Container Insights ロググループを選択します。次の例のクエリが機能するためには、ロググループ名が performance で終わる必要があります。

   ロググループを選択すると、CloudWatch Logs Insights はロググループのデータのフィールドを自動的に検出し、右側のペインの [Discovered fields (検出済みフィールド)] に表示します。また、このロググループのログイベントを時間の経過に従って棒グラフで表示します。この棒グラフは、表に示されるイベントだけでなく、クエリと時間範囲に一致するロググループ内のイベントの分布を示します。
4. クエリエディタで、デフォルトのクエリを次のクエリで置き換え、[クエリの実行] を選択します。

   ```
   STATS avg(node_cpu_utilization) as avg_node_cpu_utilization by NodeName
   | SORT avg_node_cpu_utilization DESC
   ```

   このクエリでは、ノードの平均 CPU 使用率でソートされたノードのリストが表示されます。
5. 別の例を試すには、このクエリを別のクエリに置き換え、[Run query (クエリの実行)] を選択します。その他のサンプルクエリについては、このページの後半で示します。

   ```
   STATS avg(number_of_container_restarts) as avg_number_of_container_restarts by
    PodName
   | SORT avg_number_of_container_restarts DESC
   ```

   このクエリでは、コンテナ再起動の平均数によってソートされたポッドのリストが表示されます。

6. 別のクエリを試す場合は、画面の右にあるリストのインクルードフィールドを選択できます。クエリ構文の詳細については、「CloudWatch Logs Insights クエリ構文」を参照してください。

リソースのリストを表示するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで、[Resources (リソース)] を選択します。
3. デフォルトのビューは、Container Insights によってモニターリングされているリソースと、それらのリソースに対して設定したアラームのリストです。リソースの視覚的なマップを表示するには、[Map view (マップビュー)] を選択します。
4. マップビューから、マップ内の任意のリソースの上にポインタを置くと、そのリソースに関する基本的なメトリクスを表示できます。任意のリソースを選択して、そのリソースに関するより詳細なグラフを表示できます。

ユースケース: Amazon ECS コンテナでのタスクレベルのメトリクスの表示

次の例は、CloudWatch Logs Insights を使用して Container Insights のログをより詳しく調べる方法を示しています。他の例については、ブログ記事「Amazon ECS 向け Amazon CloudWatch Container Insights のご紹介」をご覧ください。

Container Insights は、タスクレベルの詳細度でメトリクスを自動的に生成しません。次のクエリは、CPU およびメモリ使用量のタスクレベルのメトリクスを表示します。

```
stats avg(CpuUtilized) as CPU, avg(MemoryUtilized) as Mem by TaskId, ContainerName
| sort Mem, CPU desc
```

Container Insights のその他のサンプルクエリ

コンテナの再起動の平均回数順にソートされるポッドのリスト

```
STATS avg(number_of_container_restarts) as avg_number_of_container_restarts by PodName
| SORT avg_number_of_container_restarts DESC
```

リクエストされたポッドと実行中のポッドの比較

```
fields @timestamp, @message
| sort @timestamp desc
```

```
| filter Type="Pod"
| stats min(pod_number_of_containers) as requested,
 min(pod_number_of_running_containers) as running, ceil(avg(pod_number_of_containers-
pod_number_of_running_containers)) as pods_missing by kubernetes.pod_name
| sort pods_missing desc
```

クラスターノードの障害数

```
stats avg(cluster_failed_node_count) as CountOfNodeFailures
| filter Type="Cluster"
| sort @timestamp desc
```

コンテナ名別のアプリケーションログエラー

```
stats count() as countoferrors by kubernetes.container_name
| filter stream="stderr"
| sort countoferrors desc
```

Container Insights により収集されるメトリクス

Container Insights は、Amazon ECS と Amazon ECS 上の AWS Fargate の 1 つのメトリクスセットを収集し、Amazon EKS、Amazon EKS 上の AWS Fargate、および Kubernetes では別のセットを収集します。

メトリクスは、コンテナタスクがしばらく実行されるまで表示されません。

トピック

- Amazon ECS Container Insights メトリクス
- Amazon EKS および Kubernetes Container Insights のメトリクス

Amazon ECS Container Insights メトリクス

以下の表は、Container Insights で Amazon ECS に収集されるメトリクスとディメンションを示しています。これらのメトリクスは `ECS/ContainerInsights` 名前空間にあります。詳細については、「メトリクス」を参照してください。

コンソールに Container Insights メトリクスが表示されない場合は、Container Insights のセットアップが完了していることを確認します。メトリクスは、Container Insights が完全にセットアップされるまで表示されません。詳細については、「Container Insights の設定」を参照してください。

「クラスターおよびサービスレベルの Amazon ECS でメトリクスの Container Insights の設定」のステップを完了すると、以下のメトリクスが使用できるようになります

メトリクス名	ディメンション	説明
ContainerInstanceCount	ClusterName	クラスターに登録されている Amazon ECS エージェントを実行している EC2 インスタンスの数。 このメトリクスは、クラスターで Amazon ECS タスクが実行されているコンテナインスタンスに対してのみ収集されます。Amazon ECS タスクがない空のコンテナインスタンスについては収集されません。 単位: 個
CpuUtilized	TaskDefinitionFamily , ClusterName ServiceName , ClusterName ClusterName	使用しているディメンションセットによって指定されるリソースのタスクにより使用されている CPU ユニット数。 このメトリクスは、タスク定義に CPU 予約が定義されているタスクに対してのみ収集されます。 単位: なし
CpuReserved	TaskDefinitionFamily , ClusterName	使用しているディメンションセットによって指定されるリソースのタス

メトリクス名	ディメンション	説明
	ServiceName , ClusterName ClusterName	クにより予約されている CPU ユニット数。 このメトリクスは、タスク定義に CPU 予約が定義されているタスクに対してのみ収集されます。 単位: なし
DeploymentCount	ServiceName , ClusterName	Amazon ECS サービスでのデプロイの数。 単位: 個
DesiredTaskCount	ServiceName , ClusterName	Amazon ECS サービスに必要なタスクの数。 単位: 個

メトリクス名	ディメンション	説明
EBSFilesystemSize	VolumeName , TaskDefinitionFamily , ClusterName TaskDefinitionFamily , ClusterName ServiceName , ClusterName	使用しているディメンションで指定されたリソースに割り当てられる Amazon EBS ファイルシステムストレージの合計容量 (GB)。 このメトリクスは、プラットフォームバージョン 1.4.0 を使用する Fargate で実行されている Amazon ECS インフラストラクチャ、またはコンテナエージェントバージョン 1.79.0 以降を使用する Amazon EC2 インスタンスで実行されるタスクでのみ使用できます。 単位: ギガバイト (GB)

メトリクス名	ディメンション	説明
EBSFilesystemUtilized	VolumeName , TaskDefinitionFamily , ClusterName TaskDefinitionFamily , ClusterName ServiceName , ClusterName	使用しているディメンションで指定されたリソースに使用される Amazon EBS ファイルシステムストレージの合計容量 (GB)。 このメトリクスは、プラットフォームバージョン 1.4.0 を使用する Fargate で実行されている Amazon ECS インフラストラクチャ、またはコンテナエージェントバージョン 1.79.0 以降を使用する Amazon EC2 インスタンスで実行されるタスクでのみ使用できます。 Fargate で実行されるタスクの場合、Fargate は Fargate でのみ使用されるディスク上のスペースを予約します。Fargate が使用するスペースにコストは発生しませんが、df のようなツールで使用されるこの追加ストレージが表示されます。 単位: ギガバイト (GB)

メトリクス名	ディメンション	説明
EphemeralStorageReserved [1]	TaskDefinitionFamily , ClusterName ServiceName , ClusterName ClusterName	使用しているディメンションによって指定されたリソースのエフェメラルストレージから予約されたバイト数。エフェメラルストレージは、コンテナルートファイルシステム、およびコンテナイメージとタスク定義で定義されているバインドマウントホストボリュームに使用されます。エフェメラルストレージの容量は、実行中のタスクでは変更できません。 このメトリクスは、Fargate Linux プラットフォームのバージョン 1.4.0 以降で実行されるタスクでのみ使用できます。 単位: ギガバイト (GB)

メトリクス名	ディメンション	説明
EphemeralStorageUtilized [1]	TaskDefinitionFamily , ClusterName ServiceName , ClusterName ClusterName	使用しているディメンションで指定されたリソースのエフェメラルストレージから使用されたバイト数。エフェメラルストレージは、コンテナルートファイルシステム、およびコンテナイメージとタスク定義で定義されているバインドマウントホストボリュームに使用されます。エフェメラルストレージの容量は、実行中のタスクでは変更できません。 このメトリクスは、Fargate Linux プラットフォームのバージョン 1.4.0 以降で実行されるタスクでのみ使用できます。 単位: ギガバイト (GB)

メトリクス名	ディメンション	説明
MemoryUtilized	TaskDefinitionFamily , ClusterName ServiceName , ClusterName ClusterName	使用しているディメンションセットによって指定されるリソースのタスクにより使用されているメモリ。 このメトリクスは、タスク定義にメモリ予約が定義されているタスクに対してのみ収集されます。 単位: メガバイト
MemoryReserved	TaskDefinitionFamily , ClusterName ServiceName , ClusterName ClusterName	使用しているディメンションセットによって指定されるリソースのタスクにより予約されているメモリ。 このメトリクスは、タスク定義にメモリ予約が定義されているタスクに対してのみ収集されます。 単位: メガバイト

メトリクス名	ディメンション	説明
NetworkRxBytes	TaskDefinitionFamily , ClusterName ServiceName , ClusterName ClusterName	使用しているディメンションによって指定されたリソースにより受信されるバイト数。このメトリクスは、Docker ランタイムから取得されます。 このメトリクスは、awsvpc または bridge ネットワークモードを使用するタスクのコンテナでのみ使用できます。 単位: バイト/秒
NetworkTxBytes	TaskDefinitionFamily , ClusterName ServiceName , ClusterName ClusterName	使用しているディメンションによって指定されたリソースにより送信されるバイト数。このメトリクスは、Docker ランタイムから取得されます。 このメトリクスは、awsvpc または bridge ネットワークモードを使用するタスクのコンテナでのみ使用できます。 単位: バイト/秒
PendingTaskCount	ServiceName , ClusterName	現在、PENDING 状態にあるタスクの数。 単位: 個

メトリクス名	ディメンション	説明
RunningTaskCount	ServiceName , ClusterName	現在、RUNNING 状態にあるタスクの数。 単位: 個
ServiceCount	ClusterName	クラスター内のサービスの数。 単位: 個
StorageReadBytes	TaskDefinitionFamily , ClusterName ServiceName , ClusterName ClusterName	使用しているディメンションによって指定されたリソース内のインスタンスのストレージから読み取られたバイト数。これにはストレージデバイスの読み取りバイト数は含まれません。このメトリクスは、Docker ランタイムから取得されます。 単位: バイト
StorageWriteBytes	TaskDefinitionFamily , ClusterName ServiceName , ClusterName ClusterName	使用しているディメンションによって指定されたリソースのストレージに書き込まれたバイト数。このメトリクスは、Docker ランタイムから取得されます。 単位: バイト
TaskCount	ClusterName	クラスターで実行されているタスクの数。 単位: 個

メトリクス名	ディメンション	説明
TaskSetCount	ServiceName , ClusterName	サービス内のタスクセットの数。 単位: 個

> **Note**
>
> EphemeralStorageReserved および EphemeralStorageUtilized メトリクスは、Fargate Linux プラットフォームのバージョン 1.4.0 以降で実行されるタスクでの使用できます。
> Fargate はディスク上のスペースを予約します。スペースは Fargate によってのみ使用されます。これには課金されることはありません。これらのメトリクスには表示されません。ただし、この追加ストレージは、df などの他のツールでも確認できます。

「Amazon ECS で EC2 インスタンスレベルのメトリクスを収集するための CloudWatch エージェントのデプロイ」のステップを完了すると、以下のメトリクスが使用できるようになります

メトリクス名	ディメンション	説明
instance_cpu_limit	ClusterName	このクラスター内の単一の EC2 インスタンスに割り当てることができる CPU ユニットの最大数。 単位: なし
instance_cpu_reserved_capacity	ClusterName InstanceId , ContainerInstanceId , ClusterName	クラスター内の単一の EC2 インスタンスで現在予約されている CPU の割合。 単位: パーセント

メトリクス名	ディメンション	説明
instance_cpu_usage_total	ClusterName	クラスター内の単一 EC2 インスタンスで使用されている CPU ユニットの数。 単位: なし
instance_cpu_utilization	ClusterName InstanceId , ContainerInstanceId , ClusterName	クラスター内の単一の EC2 インスタンスで使用されている CPU ユニットの合計割合。 単位: パーセント
instance_filesystem_utilization	ClusterName InstanceId , ContainerInstanceId , ClusterName	クラスター内の単一の EC2 インスタンスで使用されているファイルシステム容量の合計割合。 単位: パーセント
instance_memory_limit	ClusterName	このクラスター内の単一の EC2 インスタンスに割り当てることができるメモリの最大量 (バイト単位) 。 単位: バイト

メトリクス名	ディメンション	説明
instance_memory_reserved_capacity	ClusterName InstanceId , ContainerInstanceId , ClusterName	クラスター内の単一の EC2 インスタンスで現在予約されているメモリの割合。 単位: パーセント
instance_memory_utilization	ClusterName InstanceId , ContainerInstanceId , ClusterName	クラスター内の単一の EC2 インスタンスで使用されているメモリの合計割合。 単位: パーセント
instance_memory_working_set	ClusterName	クラスター内の単一の EC2 インスタンスで使用されているメモリの量 (バイト単位) 。 単位: バイト
instance_network_total_bytes	ClusterName	クラスター内の単一の EC2 インスタンスでネットワーク上で送受信された 1 秒あたりの合計バイト数。 単位: バイト/秒
instance_number_of_running_tasks	ClusterName	クラスター内の単一の EC2 インスタンスで実行中のタスクの数。 単位: 個

Amazon EKS および Kubernetes Container Insights のメトリクス

以下の表は、Container Insights で収集する Amazon EKS および Kubernetes のメトリクスとディメンションを示しています。これらのメトリクスは ContainerInsights 名前空間にあります。詳細については、「メトリクス」を参照してください。

コンソールに Container Insights メトリクスが表示されない場合は、Container Insights のセットアップが完了していることを確認します。メトリクスは、Container Insights が完全にセットアップされるまで表示されません。詳細については、「Container Insights の設定」を参照してください。

Amazon EKS アドオンのバージョン 1.5.0 以降、または CloudWatch エージェントのバージョン 1.300035.0 を使用している場合、次の表に示されているほとんどのメトリクスは Linux ノードと Windows ノードの両方で収集されます。Windows で収集されないメトリクスを確認するには、表の「メトリクス名」列を参照してください。

Container Insights の元のバージョンでは、メトリクスはカスタムメトリクスとして課金されます。Amazon EKS 向けにオブザーバビリティが強化された Container Insights では、観察結果ごと Container Insights メトリクスに課金されます。保存されたメトリクスまたは取り込まれたログごとには課金されません。CloudWatch の料金の詳細については、Amazon CloudWatch の料金をご覧ください。

> **Note**
>
> Windows で、ホストプロセスコンテナの pod_network_rx_bytes や pod_network_tx_bytes などのネットワークメトリクスは収集されません。

メトリクス名	Container Insights の任意のバージョンのディメンション	Amazon EKS 向けにオブザーバビリティが強化された Container Insights のその他のディメンション	説明
cluster_failed_node_count	ClusterName		クラスター内の失敗したワーカーノードの数。ノードの状態に何らかの問題

メトリクス名	Container Insights の任意のバージョンのディメンション	Amazon EKS 向けにオブザーバビリティが強化された Container Insights のその他のディメンション	説明
			がある場合は、そのノードは失敗したとみなされます。詳細については、Kubernetes ドキュメントの「Conditions (状態)」を参照してください。
cluster_node_count	ClusterName		クラスター内のワーカーノードの総数。
namespace_number_of_running_pods	Namespace ClusterName ClusterName		使用しているディメンションによって指定されたリソースの名前空間ごとに実行されているポッドの数。
node_cpu_limit	ClusterName	ClusterName , InstanceId , NodeName	このクラスター内の単一のノードに割り当てることができる CPU ユニットの最大数。

メトリクス名	Container Insights の任意のバージョンのディメンション	Amazon EKS 向けにオブザーバビリティが強化された Container Insights のその他のディメンション	説明
node_cpu_reserved_capacity	NodeName, ClusterName , InstanceId ClusterName		ノードコンポーネント (kubelet、kube-proxy、Docker など) に予約されている CPU ユニットの割合。 計算式: node_cpu_request / node_cpu_limit **Note** node_cpu_request はメトリクスとして直接報告されませんが、パフォーマンスログイベント内のフィールドです。詳細については、「Amazon EKS と Kubernetes のパフォーマンスログイベントの関連フ

メトリクス名	Container Insights の任意のバージョンのディメンション	Amazon EKS 向けにオブザーバビリティが強化された Container Insights のその他のディメンション	説明
			ィールド」を参照してください。
node_cpu_usage_total	ClusterName	ClusterName , InstanceId , NodeName	クラスターのノードで使用されている CPU ユニットの数。
node_cpu_utilization	NodeName, ClusterName , InstanceId ClusterName		クラスター内のノードで使用されている CPU ユニットの合計使用率。 計算式: node_cpu_usage_total / node_cpu_limit

メトリクス名	Container Insights の任意のバージョンのディメンション	Amazon EKS 向けにオブザーバビリティが強化された Container Insights のその他のディメンション	説明
node_filesystem_utilization	NodeName, ClusterName , InstanceId ClusterName		クラスター内のノードで使用されているファイルシステム容量の合計使用率。 計算式: node_filesystem_usage / node_filesystem_capacity **Note** node_filesystem_usage および node_filesystem_capacity はメトリクスとして直接報告されませんが、パフォーマンスログイベント内のフィールドです。詳細については、「Amazon EKS と

メトリクス名	Container Insights の任意のバージョンのディメンション	Amazon EKS 向けにオブザーバビリティが強化された Container Insights のその他のディメンション	説明
			Kubernetes のパフォーマンスログイベントの関連フィールド」を参照してください。
node_memory_limit	ClusterName	ClusterName , InstanceId , NodeName	このクラスター内の単一のノードに割り当てることができるメモリの最大量 (バイト単位)。
node_filesystem_inodes このメトリクスは Amazon EKS 向けにオブザーバビリティが強化された Container Insights でのみ利用できます。Windows ではご利用いただけません。		ClusterName ClusterName , InstanceId , NodeName	ノード上の inode (使用済みおよび未使用) の総数。

メトリクス名	Container Insights の任意のバージョンのディメンション	Amazon EKS 向けにオブザーバビリティが強化された Container Insights のその他のディメンション	説明
node_filesystem_inodes_free このメトリクスは Amazon EKS 向けにオブザーバビリティが強化された Container Insights でのみ利用できます。Windows ではご利用いただけません。		ClusterName ClusterName , InstanceId , NodeName	ノード上の未使用の inode の数。

メトリクス名	Container Insights の任意のバージョンのディメンション	Amazon EKS 向けにオブザーバビリティが強化された Container Insights のその他のディメンション	説明
node_memory_reserved_capacity	NodeName, ClusterName , InstanceId ClusterName		クラスター内のノードで現在使用されているメモリの割合。 計算式: node_memory_request / node_memory_limit **Note** node_memory_request はメトリクスとして直接報告されませんが、パフォーマンスログイベント内のフィールドです。詳細については、「Amazon EKS と Kubernetes のパフォーマンスログイベントの関連フィールド」を

メトリクス名	Container Insights の任意のバージョンのディメンション	Amazon EKS 向けにオブザーバビリティが強化された Container Insights のその他のディメンション	説明
			参照してください。
node_memory_utilization	NodeName, ClusterName , InstanceId ClusterName		ノードによって現在使用されているメモリの割合。これは、ノードのメモリ制限で割られたノードのメモリ使用量の割合です。 計算式: node_memory_working_set / node_memory_limit
node_memory_working_set	ClusterName	ClusterName , InstanceId , NodeName	クラスターで現在稼働しているノードのセットで使用されているメモリの量 (バイト単位)。

メトリクス名	Container Insights の任意のバージョンのディメンション	Amazon EKS 向けにオブザーバビリティが強化された Container Insights のその他のディメンション	説明
node_network_total_bytes	NodeName, ClusterName , InstanceId ClusterName		クラスターのノードごとにネットワーク経由で送信および受信された合計バイト数。 計算式: node_network_rx_bytes + node_network_tx_bytes **Note** node_network_rx_bytes および node_network_tx_bytes はメトリクスとして直接報告されませんが、パフォーマンスログイベント内のフィールドです。詳細については、「Amazon EKS と Kubernetes

メトリクス名	Container Insights の任意のバージョンのディメンション	Amazon EKS 向けにオブザーバビリティが強化された Container Insights のその他のディメンション	説明
			のパフォーマンスログイベントの関連フィールド」を参照してください。
node_number_of_running_containers	NodeName, ClusterName , InstanceId ClusterName		クラスターのノードごとに実行中のコンテナの数。
node_number_of_running_pods	NodeName, ClusterName , InstanceId ClusterName		クラスターのノードごとに実行中のポッドの数。
node_status_allocatable_pods このメトリクスは Amazon EKS 向けにオブザーバビリティが強化された Container Insights でのみ利用できます		ClusterName ClusterName , InstanceId , NodeName	割り当て可能なリソースに基づいてノードに割り当てることができるポッドの数。システムデーモンの予約とハードエビクションのしきい値の考慮後のノードの残りの容量として定義されます。

メトリクス名	Container Insights の任意のバージョンのディメンション	Amazon EKS 向けにオブザーバビリティが強化された Container Insights のその他のディメンション	説明
node_status_capacity_pods このメトリクスは Amazon EKS 向けにオブザーバビリティが強化された Container Insights でのみ利用できます		ClusterName ClusterName , InstanceId , NodeName	容量に基づいてノードに割り当てることができるポッドの数。
node_status_condition_ready このメトリクスは Amazon EKS 向けにオブザーバビリティが強化された Container Insights でのみ利用できます		ClusterName ClusterName , InstanceId , NodeName	ノードステータスの条件 Ready が正しいかどうかを示します。

メトリクス名	Container Insights の任意のバージョンのディメンション	Amazon EKS 向けにオブザーバビリティが強化された Container Insights のその他のディメンション	説明
node_status_condition_memory_pressure このメトリクスは Amazon EKS 向けにオブザーバビリティが強化された Container Insights でのみ利用できます		ClusterName ClusterName , InstanceId , NodeName	ノードステータスの条件 MemoryPressure が正しいかどうかを示します。
node_status_condition_pid_pressure このメトリクスは Amazon EKS 向けにオブザーバビリティが強化された Container Insights でのみ利用できます		ClusterName ClusterName , InstanceId , NodeName	ノードステータスの条件 PIDPressure が正しいかどうかを示します。

メトリクス名	Container Insights の任意のバージョンのディメンション	Amazon EKS 向けにオブザーバビリティが強化された Container Insights のその他のディメンション	説明
node_status_condition_disk_pressure このメトリクスは Amazon EKS 向けにオブザーバビリティが強化された Container Insights でのみ利用できます		ClusterName ClusterName , InstanceId , NodeName	ノードステータスの条件 OutOfDisk が正しいかどうかを示します。
node_status_condition_unknown このメトリクスは Amazon EKS 向けにオブザーバビリティが強化された Container Insights でのみ利用できます		ClusterName ClusterName , InstanceId , NodeName	ノードステータス条件のいずれかが不明であるかどうかを示します。

メトリクス名	Container Insights の任意のバージョンのディメンション	Amazon EKS 向けにオブザーバビリティが強化された Container Insights のその他のディメンション	説明
node_interface_network_rx_dropped このメトリクスは Amazon EKS 向けにオブザーバビリティが強化された Container Insights でのみ利用できます このメトリクスは Amazon EKS 向けにオブザーバビリティが強化された Container Insights でのみ利用できます		ClusterName ClusterName , InstanceId , NodeName	ノード上のネットワークインターフェイスによって受信されたが、その後削除されたパケットの数。
node_interface_network_tx_dropped このメトリクスは Amazon EKS 向けにオブザーバビリティが強化された Container Insights でのみ利用できます		ClusterName ClusterName , InstanceId , NodeName	ノード上のネットワークインターフェイスによって送信される予定だったが、削除されたパケットの数。

メトリクス名	Container Insights の任意のバージョンのディメンション	Amazon EKS 向けにオブザーバビリティが強化された Container Insights のその他のディメンション	説明
node_diskio_io_service_bytes_total このメトリクスは Amazon EKS 向けにオブザーバビリティが強化された Container Insights でのみ利用できます。Windows ではご利用いただけません。		ClusterName ClusterName , InstanceId , NodeName	ノード上のすべての I/O オペレーションにより送信された合計バイト数。
node_diskio_io_serviced_total このメトリクスは Amazon EKS 向けにオブザーバビリティが強化された Container Insights でのみ利用できます。Windows ではご利用いただけません。		ClusterName ClusterName , InstanceId , NodeName	ノード上の I/O オペレーションの総数。

メトリクス名	Container Insights の任意のバージョンのディメンション	Amazon EKS 向けにオブザーバビリティが強化された Container Insights のその他のディメンション	説明
pod_cpu_reserved_capacity	PodName、名前空間、ClusterName ClusterName	ClusterName , Namespace , PodName, FullPodName ClusterName , Namespace , Service	クラスターのポッドごとに予約されている CPU 容量。 計算式: pod_cpu_request / node_cpu_limit **Note** pod_cpu_request はメトリクスとして直接報告されませんが、パフォーマンスログイベント内のフィールドです。詳細については、「Amazon EKS と Kubernetes のパフォーマンスログイベントの関連フィールド」を

メトリクス名	Container Insights の任意のバージョンのディメンション	Amazon EKS 向けにオブザーバビリティが強化された Container Insights のその他のディメンション	説明
			参照してください。

メトリクス名	Container Insights の任意のバージョンのディメンション	Amazon EKS 向けにオブザーバビリティが強化された Container Insights のその他のディメンション	説明
pod_cpu_utilization	PodName、名前空間、ClusterName 名前空間: ClusterName Service、名前空間、ClusterName ClusterName	ClusterName , Namespace , PodName, FullPodName	ポッドで使用されている CPU ユニットの割合。 計算式: pod_cpu_usage_total / node_cpu_limit **Note** pod_cpu_usage_total はメトリクスとして直接報告されませんが、パフォーマンスログイベント内のフィールドです。詳細については、「Amazon EKS と Kubernetes のパフォーマンスログイベントの関連フィールド」を

メトリクス名	Container Insights の任意のバージョンのディメンション	Amazon EKS 向けにオブザーバビリティが強化された Container Insights のその他のディメンション	説明
			参照してください。

メトリクス名	Container Insights の任意のバージョンのディメンション	Amazon EKS 向けにオブザーバビリティが強化された Container Insights のその他のディメンション	説明
pod_cpu_utilization_over_pod_limit	PodName、名前空間、ClusterName 名前空間: ClusterName Service、名前空間、ClusterName ClusterName	ClusterName , Namespace , PodName, FullPodName	ポッドの制限に対する、ポッドで使用されている CPU ユニットの割合。 計算式: pod_cpu_usage_total / pod_cpu_limit **Note** pod_cpu_usage_total および pod_cpu_limit はメトリクスとして直接報告されませんが、パフォーマンスログイベント内のフィールドです。詳細については、「Amazon EKS と Kubernetes のパフォーマンスログイベントの関連フ

メトリクス名	Container Insights の任意のバージョンのディメンション	Amazon EKS 向けにオブザーバビリティが強化された Container Insights のその他のディメンション	説明
			ィールド」を参照してください。

メトリクス名	Container Insights の任意のバージョンのディメンション	Amazon EKS 向けにオブザーバビリティが強化された Container Insights のその他のディメンション	説明
pod_memory_reserved_capacity	PodName、名前空間、ClusterName ClusterName	ClusterName , Namespace , PodName, FullPodName ClusterName , Namespace , Service	ポッド用に予約されているメモリの割合。 計算式: pod_memory_request / node_memory_limit **Note** pod_memory_request はメトリクスとして直接報告されませんが、パフォーマンスログイベント内のフィールドです。詳細については、「Amazon EKS と Kubernetes のパフォーマンスログイベントの関連フィールド」を

メトリクス名	Container Insights の任意のバージョンのディメンション	Amazon EKS 向けにオブザーバビリティが強化された Container Insights のその他のディメンション	説明
			参照してください。

メトリクス名	Container Insights の任意のバージョンのディメンション	Amazon EKS 向けにオブザーバビリティが強化された Container Insights のその他のディメンション	説明
pod_memory_utilization	PodName、名前空間、ClusterName 名前空間: ClusterName Service、名前空間、ClusterName ClusterName	ClusterName , Namespace , PodName, FullPodName	ポッドが現在使用しているメモリの割合。 計算式: pod_memory_working_set / node_memory_limit **Note** pod_memory_working_set はメトリクスとして直接報告されませんが、パフォーマンスログイベント内のフィールドです。詳細については、「Amazon EKS と Kubernetes のパフォーマンスログイベントの関連フィールド」を

メトリクス名	Container Insights の任意のバージョンのディメンション	Amazon EKS 向けにオブザーバビリティが強化された Container Insights のその他のディメンション	説明
			参照してください。

メトリクス名	Container Insights の任意のバージョンのディメンション	Amazon EKS 向けにオブザーバビリティが強化された Container Insights のその他のディメンション	説明
pod_memory_utilization_over_pod_limit	PodName、名前空間、ClusterName 名前空間: ClusterName Service、名前空間、ClusterName ClusterName	ClusterName , Namespace , PodName, FullPodName	ポッドの制限に対する、ポッドで使用されているメモリの割合。ポッドのいずれかのコンテナに、定義されたメモリ制限がない場合、このメトリクスは表示されません。 計算式: pod_memory_working_set / pod_memory_limit **Note** pod_memory_working_set はメトリクスとして直接報告されませんが、パフォーマンスログイベント内のフィールドです。詳細については、「Amazon

メトリクス名	Container Insights の任意のバージョンのディメンション	Amazon EKS 向けにオブザーバビリティが強化された Container Insights のその他のディメンション	説明
			EKS と Kubernetes のパフォーマンスログイベントの関連フィールド」を参照してください。

メトリクス名	Container Insights の任意のバージョンのディメンション	Amazon EKS 向けにオブザーバビリティが強化された Container Insights のその他のディメンション	説明
pod_network_rx_bytes	PodName、名前空間、ClusterName 名前空間: ClusterName Service、名前空間、ClusterName ClusterName	ClusterName , Namespace , PodName, FullPodName	ポッドによって、ネットワーク経由で受信されているバイト数。 計算式: sum(pod_interface_network_rx_bytes) **Note** pod_interface_network_rx_bytes はメトリクスとして直接報告されませんが、パフォーマンスログイベント内のフィールドです。詳細については、「Amazon EKS と Kubernetes のパフォーマンスログイベントの関連フ

メトリクス名	Container Insights の任意のバージョンのディメンション	Amazon EKS 向けにオブザーバビリティが強化された Container Insights のその他のディメンション	説明
			ィールド」を参照してください。

メトリクス名	Container Insights の任意のバージョンのディメンション	Amazon EKS 向けにオブザーバビリティが強化された Container Insights のその他のディメンション	説明
pod_network_tx_bytes	PodName、名前空間、ClusterName 名前空間: ClusterName Service、名前空間、ClusterName ClusterName	ClusterName , Namespace , PodName, FullPodName	ポッドによって、ネットワーク経由で送信されているバイト数。 計算式: sum(pod_interface_network_tx_bytes) **Note** pod_interface_network_tx_bytes はメトリクスとして直接報告されませんが、パフォーマンスログイベント内のフィールドです。詳細については、「Amazon EKS と Kubernetes のパフォーマンスログイベントの関連フ

メトリクス名	Container Insights の任意のバージョンのディメンション	Amazon EKS 向けにオブザーバビリティが強化された Container Insights のその他のディメンション	説明
			ィールド」を参照してください。

メトリクス名	Container Insights の任意のバージョンのディメンション	Amazon EKS 向けにオブザーバビリティが強化された Container Insights のその他のディメンション	説明

`pod_cpu_request`
 	
このメトリクスは Amazon EKS 向けにオブザーバビリティが強化された Container Insights でのみ利用できます
		`ClusterName`
		
		`PodName`, `Namespace` , `ClusterName`
		
		`Namespace` , `ClusterName` , `Service`
		
		`ClusterName` , `Namespace` , `PodName`, `FullPodName`
			ポッドの CPU リクエスト。
			
			計算式: `sum(container_cpu_request)`
			
			Note
			`pod_cpu_request` はメトリクスとして直接報告されませんが、パフォーマンスログイベント内のフィールドです。詳細については、「Amazon EKS と Kubernetes のパフォーマンスログイベントの関連フィールド」を参照してください。

メトリクス名	Container Insights の任意のバージョンのディメンション	Amazon EKS 向けにオブザーバビリティが強化された Container Insights のその他のディメンション	説明
pod_memory_request このメトリクスは Amazon EKS 向けにオブザーバビリティが強化された Container Insights でのみ利用できます		ClusterName PodName, Namespace , ClusterName Namespace , ClusterName , Service ClusterName , Namespace , PodName, FullPodName	ポッドのメモリリクエスト。 計算式: sum(container_memory_request) **Note** pod_memory_request はメトリクスとして直接報告されませんが、パフォーマンスログイベント内のフィールドです。詳細については、「Amazon EKS と Kubernetes のパフォーマンスログイベントの関連フィールド」を参照してください。

メトリクス名	Container Insights の任意のバージョンのディメンション	Amazon EKS 向けにオブザーバビリティが強化された Container Insights のその他のディメンション	説明

`pod_cpu_limit`
 	
このメトリクスは Amazon EKS 向けにオブザーバビリティが強化された Container Insights でのみ利用できます
		`ClusterName`
		
		`PodName`, `Namespace` , `ClusterName`
		
		`Namespace` , `ClusterName` , `Service`
		
		`ClusterName` , `Namespace` , `PodName`, `FullPodName`
			ポッド内のコンテナに定義されている CPU 制限。ポッド内のいずれかのコンテナに定義された CPU 制限がない場合、このメトリクスは表示されません。
			
			計算式: `sum(container_cpu_limit)`
			
			Note
			`pod_cpu_limit` はメトリクスとして直接報告されませんが、パフォーマンスログイベント内のフィールドです。詳細については、「Amazon EKS と Kubernetes のパフォーマンスログイベ

メトリクス名	Container Insights の任意のバージョンのディメンション	Amazon EKS 向けにオブザーバビリティが強化された Container Insights のその他のディメンション	説明
			ントの関連フィールド」を参照してください。

メトリクス名	Container Insights の任意のバージョンのディメンション	Amazon EKS 向けにオブザーバビリティが強化された Container Insights のその他のディメンション	説明
pod_memory_limit このメトリクスは Amazon EKS 向けにオブザーバビリティが強化された Container Insights でのみ利用できます		ClusterName PodName, Namespace , ClusterName Namespace , ClusterName , Service ClusterName , Namespace , PodName, FullPodName	ポッド内のコンテナに定義されているメモリ上限。ポッドのいずれかのコンテナに、定義されたメモリ制限がない場合、このメトリクスは表示されません。 計算式: sum(container_memory_limit) **Note** pod_cpu_limit はメトリクスとして直接報告されませんが、パフォーマンスログイベント内のフィールドです。詳細については、「Amazon EKS と Kubernetes のパフォーマンスログイベ

メトリクス名	Container Insights の任意のバージョンのディメンション	Amazon EKS 向けにオブザーバビリティが強化された Container Insights のその他のディメンション	説明
			ントの関連フィールド」を参照してください。
pod_status_failed このメトリクスは Amazon EKS 向けにオブザーバビリティが強化された Container Insights でのみ利用できます		ClusterName PodName, Namespace , ClusterName Namespace , ClusterName , Service ClusterName , Namespace , PodName, FullPodName	ポッド内のすべてのコンテナが終了したことを示します。また、少なくとも 1 つのコンテナがゼロ以外のステータスで終了したか、システムによって終了されたことを示します。

メトリクス名	Container Insights の任意のバージョンのディメンション	Amazon EKS 向けにオブザーバビリティが強化された Container Insights のその他のディメンション	説明
pod_status_ready このメトリクスは Amazon EKS 向けにオブザーバビリティが強化された Container Insights でのみ利用できます		ClusterName PodName, Namespace , ClusterName Namespace , ClusterName , Service ClusterName , Namespace , PodName, FullPodName	ポッド内のすべてのコンテナの準備が整い、条件 Container Ready に達したことを示します。
pod_status_running このメトリクスは Amazon EKS 向けにオブザーバビリティが強化された Container Insights でのみ利用できます		ClusterName PodName, Namespace , ClusterName Namespace , ClusterName , Service ClusterName , Namespace , PodName, FullPodName	ポッド内のすべてのコンテナが実行中であることを示します。

メトリクス名	Container Insights の任意のバージョンのディメンション	Amazon EKS 向けにオブザーバビリティが強化された Container Insights のその他のディメンション	説明
pod_status_scheduled このメトリクスは Amazon EKS 向けにオブザーバビリティが強化された Container Insights でのみ利用できます		ClusterName PodName, Namespace , ClusterName Namespace , ClusterName , Service ClusterName , Namespace , PodName, FullPodName	ポッドがノードにスケジュールされていることを示します。
pod_status_unknown このメトリクスは Amazon EKS 向けにオブザーバビリティが強化された Container Insights でのみ利用できます		ClusterName PodName, Namespace , ClusterName Namespace , ClusterName , Service ClusterName , Namespace , PodName, FullPodName	ポッドのステータスを取得できないことを示します。

メトリクス名	Container Insights の任意のバージョンのディメンション	Amazon EKS 向けにオブザーバビリティが強化された Container Insights のその他のディメンション	説明
pod_status_pending このメトリクスは Amazon EKS 向けにオブザーバビリティが強化された Container Insights でのみ利用できます		ClusterName PodName, Namespace , ClusterName Namespace , ClusterName , Service ClusterName , Namespace , PodName, FullPodName	ポッドはクラスターに受け入れられたが、1 つ以上のコンテナの準備がまだ整っていないことを示します。
pod_status_succeeded このメトリクスは Amazon EKS 向けにオブザーバビリティが強化された Container Insights でのみ利用できます		ClusterName PodName, Namespace , ClusterName Namespace , ClusterName , Service ClusterName , Namespace , PodName, FullPodName	ポッド内のすべてのコンテナが正常に終了し、再起動されていないことを示します。

メトリクス名	Container Insights の任意のバージョンのディメンション	Amazon EKS 向けにオブザーバビリティが強化された Container Insights のその他のディメンション	説明
pod_number_of_containers このメトリクスは Amazon EKS 向けにオブザーバビリティが強化された Container Insights でのみ利用できます		ClusterName PodName, Namespace , ClusterName Namespace , ClusterName , Service ClusterName , Namespace , PodName, FullPodName	ポッドの仕様で定義されているコンテナの数をレポートします。
pod_number_of_running_containers このメトリクスは Amazon EKS 向けにオブザーバビリティが強化された Container Insights でのみ利用できます		ClusterName PodName, Namespace , ClusterName Namespace , ClusterName , Service ClusterName , Namespace , PodName, FullPodName	現在、Running 状態にあるポッド内のコンテナの数をレポートします。

メトリクス名	Container Insights の任意のバージョンのディメンション	Amazon EKS 向けにオブザーバビリティが強化された Container Insights のその他のディメンション	説明
pod_container_status_terminated このメトリクスは Amazon EKS 向けにオブザーバビリティが強化された Container Insights でのみ利用できます		ClusterName PodName, Namespace , ClusterName Namespace , ClusterName , Service ClusterName , Namespace , PodName, FullPodName	Terminated 状態にあるポッド内のコンテナの数をレポートします。
pod_container_status_running このメトリクスは Amazon EKS 向けにオブザーバビリティが強化された Container Insights でのみ利用できます		ClusterName PodName, Namespace , ClusterName Namespace , ClusterName , Service ClusterName , Namespace , PodName, FullPodName	Running 状態にあるポッド内のコンテナの数をレポートします。

メトリクス名	Container Insights の任意のバージョンのディメンション	Amazon EKS 向けにオブザーバビリティが強化された Container Insights のその他のディメンション	説明
pod_container_status_waiting このメトリクスは Amazon EKS 向けにオブザーバビリティが強化された Container Insights でのみ利用できます		ClusterName PodName, Namespace , ClusterName Namespace , ClusterName , Service ClusterName , Namespace , PodName, FullPodName	Waiting 状態にあるポッド内のコンテナの数をレポートします。
pod_interface_network_rx_dropped このメトリクスは Amazon EKS 向けにオブザーバビリティが強化された Container Insights でのみ利用できます		ClusterName PodName, Namespace , ClusterName Namespace , ClusterName , Service ClusterName , Namespace , PodName, FullPodName	ポッド用にネットワークインターフェイスが受信されたが、その後削除されたパケットの数。

メトリクス名	Container Insights の任意のバージョンのディメンション	Amazon EKS 向けにオブザーバビリティが強化された Container Insights のその他のディメンション	説明
pod_interface_network_tx_dropped このメトリクスは Amazon EKS 向けにオブザーバビリティが強化された Container Insights でのみ利用できます		ClusterName PodName, Namespace , ClusterName Namespace , ClusterName , Service ClusterName , Namespace , PodName, FullPodName	ポッド用に送信される予定だったが、削除されたパケットの数。

メトリクス名	Container Insights の任意のバージョンのディメンション	Amazon EKS 向けにオブザーバビリティが強化された Container Insights のその他のディメンション	説明

`container_cpu_utilization`
 	
このメトリクスは Amazon EKS 向けにオブザーバビリティが強化された Container Insights でのみ利用できます
		`ClusterName`
		
		`PodName`, `Namespace` , `ClusterName` , `ContainerName`
		
		`PodName`, `Namespace` , `ClusterName` , `ContainerName` , `FullPodName`
			コンテナで使用されている CPU ユニットの割合。
			
			計算式: `container_cpu_usage_total / node_cpu_limit`
			
			Note
			`container_cpu_utilization` はメトリクスとして直接報告されませんが、パフォーマンスログイベント内のフィールドです。詳細については、「Amazon EKS と Kubernetes のパフォーマンスログイベントの関連フィールド」を

メトリクス名	Container Insights の任意のバージョンのディメンション	Amazon EKS 向けにオブザーバビリティが強化された Container Insights のその他のディメンション	説明
			参照してください。

メトリクス名	Container Insights の任意のバージョンのディメンション	Amazon EKS 向けにオブザーバビリティが強化された Container Insights のその他のディメンション	説明
container_cpu_utilization_over_container_limit このメトリクスは Amazon EKS 向けにオブザーバビリティが強化された Container Insights でのみ利用できます		ClusterName PodName, Namespace , ClusterName , ContainerName PodName, Namespace , ClusterName , ContainerName , FullPodName	コンテナの制限に対する、コンテナで使用されている CPU ユニットの割合。コンテナに定義された CPU 制限がない場合、このメトリクスは表示されません。 計算式: container_cpu_usage_total / container_cpu_limit **Note** container_cpu_utilization_over_container_limit はメトリクスとして直接報告されませんが、パフォーマンスログイベント内のフィールドです。詳細

メトリクス名	Container Insights の任意のバージョンのディメンション	Amazon EKS 向けにオブザーバビリティが強化された Container Insights のその他のディメンション	説明
			については、「Amazon EKS と Kubernetes のパフォーマンスログイベントの関連フィールド」を参照してください。

メトリクス名	Container Insights の任意のバージョンのディメンション	Amazon EKS 向けにオブザーバビリティが強化された Container Insights のその他のディメンション	説明

`container_memory_utilization`
 	
このメトリクスは Amazon EKS 向けにオブザーバビリティが強化された Container Insights でのみ利用できます
		`ClusterName`
		
		`PodName`, `Namespace` , `ClusterName` , `ContainerName`
		
		`PodName`, `Namespace` , `ClusterName` , `ContainerName` , `FullPodName`
			コンテナで使用されているメモリユニットの割合。
			
			計算式: `container_memory_working_set / node_memory_limit`
			
			Note
			`container_memory_utilization` はメトリクスとして直接報告されませんが、パフォーマンスログイベント内のフィールドです。詳細については、「Amazon EKS と Kubernetes のパフォーマンスログイベントの関連フ

メトリクス名	Container Insights の任意のバージョンのディメンション	Amazon EKS 向けにオブザーバビリティが強化された Container Insights のその他のディメンション	説明
			ィールド」を参照してください。

メトリクス名	Container Insights の任意のバージョンのディメンション	Amazon EKS 向けにオブザーバビリティが強化された Container Insights のその他のディメンション	説明
container_memory_utilization_over_container_limit このメトリクスは Amazon EKS 向けにオブザーバビリティが強化された Container Insights でのみ利用できます		ClusterName PodName, Namespace , ClusterName , ContainerName PodName, Namespace , ClusterName , ContainerName , FullPodName	コンテナの制限に対する、コンテナで使用されているメモリユニットの割合。コンテナに定義されたメモリ制限がない場合、このメトリクスは表示されません。 計算式: container_memory_working_set / container_memory_limit **Note** container_memory_utilization_over_container_limit はメトリクスとして直接報告されませんが、パフォーマンスログイベント内のフィールドです。詳細

メトリクス名	Container Insights の任意のバージョンのディメンション	Amazon EKS 向けにオブザーバビリティが強化された Container Insights のその他のディメンション	説明
			については、「Amazon EKS と Kubernetes のパフォーマンスログイベントの関連フィールド」を参照してください。
container_memory_failures_total このメトリクスは Amazon EKS 向けにオブザーバビリティが強化された Container Insights でのみ利用できます。Windows ではご利用いただけません。		ClusterName PodName, Namespace , ClusterName , ContainerName PodName, Namespace , ClusterName , ContainerName , FullPodName	コンテナがメモリの割り当てに失敗した回数。
pod_number_of_container_restarts	PodName、Namespace 、ClusterName		ポッドでのコンテナ再起動の合計数。

メトリクス名	Container Insights の任意のバージョンのディメンション	Amazon EKS 向けにオブザーバビリティが強化された Container Insights のその他のディメンション	説明
service_number_of_running_pods	Service、Namespace、ClusterName ClusterName		クラスターでサービス (1 つまたは複数) を実行しているポッドの数。
replicas_desired このメトリクスは Amazon EKS 向けにオブザーバビリティが強化された Container Insights でのみ利用できます		ClusterName PodName, Namespace , ClusterName	ワークロードの仕様で定義されているワークロードに必要なポッドの数。
replicas_ready このメトリクスは Amazon EKS 向けにオブザーバビリティが強化された Container Insights でのみ利用できます		ClusterName PodName, Namespace , ClusterName	準備完了の状態に達したワークロードにおけるポッドの数。

メトリクス名	Container Insights の任意のバージョンのディメンション	Amazon EKS 向けにオブザーバビリティが強化された Container Insights のその他のディメンション	説明
status_replicas_available このメトリクスは Amazon EKS 向けにオブザーバビリティが強化された Container Insights でのみ利用できます		ClusterName PodName, Namespace , ClusterName	1 つのワークロードで使用可能なポッドの数。ポッドは、ワークロードの仕様で定義されている minReadySeconds の準備が完了した時点で利用可能になります。
status_replicas_unavailable このメトリクスは Amazon EKS 向けにオブザーバビリティが強化された Container Insights でのみ利用できます		ClusterName PodName, Namespace , ClusterName	ワークロードで利用できないポッドの数。ポッドは、ワークロードの仕様で定義されている minReadySeconds の準備が完了した時点で利用可能になります。この基準を満たしていない場合、ポッドは使用できません。

メトリクス名	Container Insights の任意のバージョンのディメンション	Amazon EKS 向けにオブザーバビリティが強化された Container Insights のその他のディメンション	説明
apiserver_storage_objects このメトリクスは Amazon EKS 向けにオブザーバビリティが強化された Container Insights でのみ利用できます		ClusterName ClusterName , resource	最後の確認時に etcd に保存されたオブジェクトの数。
apiserver_request_total このメトリクスは Amazon EKS 向けにオブザーバビリティが強化された Container Insights でのみ利用できます		ClusterName ClusterName , code, verb	Kubernetes API サーバーに転送された API リクエストの総数。

メトリクス名	Container Insights の任意のバージョンのディメンション	Amazon EKS 向けにオブザーバビリティが強化された Container Insights のその他のディメンション	説明
apiserver_request_duration_seconds このメトリクスは Amazon EKS 向けにオブザーバビリティが強化された Container Insights でのみ利用できます		ClusterName ClusterName , verb	Kubernetes API サーバーへの API リクエストの応答レイテンシー。
apiserver_admission_controller_admission_duration_seconds このメトリクスは Amazon EKS 向けにオブザーバビリティが強化された Container Insights でのみ利用できます		ClusterName ClusterName , operation	アドミッションコントローラーのレイテンシー (秒単位)。アドミッションコントローラーは Kubernetes API サーバーへのリクエストをインターセプトするコードです。

メトリクス名	Container Insights の任意のバージョンのディメンション	Amazon EKS 向けにオブザーバビリティが強化された Container Insights のその他のディメンション	説明
rest_client_request_duration_seconds このメトリクスは Amazon EKS 向けにオブザーバビリティが強化された Container Insights でのみ利用できます		ClusterName ClusterName , operation	Kubernetes API サーバーを呼び出すクライアントに起こる応答レイテンシー。このメトリクスは実験段階で、Kubernetes の今後のリリースで変更される可能性があります。
rest_client_requests_total このメトリクスは Amazon EKS 向けにオブザーバビリティが強化された Container Insights でのみ利用できます		ClusterName ClusterName , code, method	クライアントにより作成された Kubernetes API サーバーへの API リクエストの総数。このメトリクスは実験段階で、Kubernetes の今後のリリースで変更される可能性があります。

メトリクス名	Container Insights の任意のバージョンのディメンション	Amazon EKS 向けにオブザーバビリティが強化された Container Insights のその他のディメンション	説明

`etcd_request_duration_seconds`
 	
このメトリクスは Amazon EKS 向けにオブザーバビリティが強化された Container Insights でのみ利用できます
		`ClusterName`
		
		`ClusterName`, `operation`
			Etcd への API 呼び出しの応答レイテンシー。このメトリクスは実験段階で、Kubernetes の今後のリリースで変更される可能性があります。

`apiserver_storage_size_bytes`
 	
このメトリクスは Amazon EKS 向けにオブザーバビリティが強化された Container Insights でのみ利用できます
		`ClusterName`
		
		`ClusterName`, `endpoint`
			物理的に割り当てられたストレージデータベースファイルのサイズ (バイト単位)。このメトリクスは実験段階で、Kubernetes の今後のリリースで変更される可能性があります。

メトリクス名	Container Insights の任意のバージョンのディメンション	Amazon EKS 向けにオブザーバビリティが強化された Container Insights のその他のディメンション	説明
apiserver_longrunning_requests このメトリクスは Amazon EKS 向けにオブザーバビリティが強化された Container Insights でのみ利用できます		ClusterName ClusterName , resource	Kubernetes API サーバーへの長時間稼働リクエストの数。
apiserver_current_inflight_requests このメトリクスは Amazon EKS 向けにオブザーバビリティが強化された Container Insights でのみ利用できます		ClusterName ClusterName , request_kind	Kubernetes API サーバーにより処理されているリクエストの数。

メトリクス名	Container Insights の任意のバージョンのディメンション	Amazon EKS 向けにオブザーバビリティが強化された Container Insights のその他のディメンション	説明
apiserver_admission_webhook_admission_duration_seconds このメトリクスは Amazon EKS 向けにオブザーバビリティが強化された Container Insights でのみ利用できます		ClusterName ClusterName , name	アドミッションウェブフックのレイテンシー (秒単位)。アドミッションウェブフックはアドミッションリクエストを受信し、何らかの処理を行う HTTP コールバックです。
apiserver_admission_step_admission_duration_seconds このメトリクスは Amazon EKS 向けにオブザーバビリティが強化された Container Insights でのみ利用できます		ClusterName ClusterName , operation	アドミッションのサブステップのレイテンシー (秒単位)。

メトリクス名	Container Insights の任意のバージョンのディメンション	Amazon EKS 向けにオブザーバビリティが強化された Container Insights のその他のディメンション	説明
apiserver_requested_deprecated_apis このメトリクスは Amazon EKS 向けにオブザーバビリティが強化された Container Insights でのみ利用できます		ClusterName ClusterName , group	Kubernetes API サーバー上での非推奨の API へのリクエストの数。
apiserver_request_total_5XX このメトリクスは Amazon EKS 向けにオブザーバビリティが強化された Container Insights でのみ利用できます		ClusterName ClusterName , code, verb	Kubernetes API サーバーへのリクエストのうち、5XX HTTP レスポンスコードで応答されたリクエストの数。

メトリクス名	Container Insights の任意のバージョンのディメンション	Amazon EKS 向けにオブザーバビリティが強化された Container Insights のその他のディメンション	説明
apiserver_storage_list_duration_seconds このメトリクスは Amazon EKS 向けにオブザーバビリティが強化された Container Insights でのみ利用できます		ClusterName ClusterName , resource	Etcd からのオブジェクトを一覧表示する応答レイテンシー。このメトリクスは実験段階で、Kubernetes の今後のリリースで変更される可能性があります。
apiserver_current_inqueue_requests このメトリクスは Amazon EKS 向けにオブザーバビリティが強化された Container Insights でのみ利用できます		ClusterName ClusterName , request_kind	Kubernetes API サーバーによりキューに保存されたリクエストの数。このメトリクスは実験段階で、Kubernetes の今後のリリースで変更される可能性があります。

メトリクス名	Container Insights の任意のバージョンのディメンション	Amazon EKS 向けにオブザーバビリティが強化された Container Insights のその他のディメンション	説明

`apiserver_flowcontrol_rejected_requests_total`
 	
このメトリクスは Amazon EKS 向けにオブザーバビリティが強化された Container Insights でのみ利用できます
		`ClusterName`
		
		`ClusterName , reason`
			API Priority and Fairness のサブシステムによって拒否されたリクエストの数。このメトリクスは実験段階で、Kubernetes の今後のリリースで変更される可能性があります。

NVIDIA GPU メトリクス

CloudWatch エージェントのバージョン `1.300034.0` 以降で Amazon EKS 向けにオブザーバビリティが強化された Container Insights は、デフォルトで EKS ワークロードから NVIDIA GPU メトリクスを収集します。CloudWatch エージェントは、CloudWatch Observability EKS アドオンのバージョン `v1.3.0-eksbuild.1` 以降を使用してインストールする必要があります。詳細については、「Amazon CloudWatch Observability EKS アドオンを使用して CloudWatch エージェントをインストールする」を参照してください。収集されるこれらの NVIDIA GPU メトリクスは、このセクションにある表に一覧表示されています。

Container Insights が NVIDIA GPU メトリクスを収集するには、次の前提条件を満たす必要があります。

- Amazon EKS 向けにオブザーバビリティが強化された Container Insights を使用するには、Amazon CloudWatch Observability EKS アドオンバージョン `v1.3.0-eksbuild.1` 以降を使用する必要があります。

- クラスターに Kubernetes 用 NVIDIA デバイスプラグインがインストールされている必要があります。
- クラスターのノードに NVIDIA コンテナツールキットがインストールされている必要があります。例えば、Amazon EKS-Optimized Accelerated AMI は、必要なコンポーネントで作成されています。

最初の CloudWatch エージェント設定ファイルの `accelerated_compute_metrics` オプションを `false` に設定することで、NVIDIA GPU メトリクスの収集をオプトアウトできます。詳細とオプトアウトの設定の例については、「(オプション) その他の設定」を参照してください。

メトリクス名	ディメンション	説明
container_gpu_memory_total	ClusterName ClusterName , Namespace , PodName, ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName , GpuDevice	コンテナに割り当てられた GPU のフレームバッファの合計サイズ (バイト)。
container_gpu_memory_used	ClusterName ClusterName , Namespace , PodName, ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName , GpuDevice	コンテナに割り当てられた GPU で使用されているフレームバッファのバイト数。

メトリクス名	ディメンション	説明
container_gpu_memory_utilization	ClusterName ClusterName , Namespace , PodName, ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName , GpuDevice	コンテナに割り当てられた GPU のうち、使用されているフレームバッファの割合。
container_gpu_power_draw	ClusterName ClusterName , Namespace , PodName, ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName , GpuDevice	コンテナに割り当てられた GPU の電力使用量 (ワット)。

メトリクス名	ディメンション	説明
container_gpu_temperature	ClusterName ClusterName , Namespace , PodName, ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName , GpuDevice	コンテナに割り当てられた GPU の温度 (摂氏)。
container_gpu_utilization	ClusterName ClusterName , Namespace , PodName, ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName , GpuDevice	コンテナに割り当てられた GPU の使用率。
node_gpu_memory_total	ClusterName ClusterName , InstanceId , NodeName ClusterName , InstanceId , InstanceType , NodeName, GpuDevice	ノードに割り当てられた GPU のフレームバッファの合計サイズ (バイト)。

メトリクス名	ディメンション	説明
node_gpu_memory_used	ClusterName ClusterName , InstanceId , NodeName ClusterName , InstanceId , InstanceType , NodeName, GpuDevice	ノードに割り当てられた GPU で使用されているフレームバッファのバイト数。
node_gpu_memory_utilization	ClusterName ClusterName , InstanceId , NodeName ClusterName , InstanceId , InstanceType , NodeName, GpuDevice	ノードに割り当てられた GPU で使用されているフレームバッファの割合。
node_gpu_power_draw	ClusterName ClusterName , InstanceId , NodeName ClusterName , InstanceId , InstanceType , NodeName, GpuDevice	ノードに割り当てられた GPU の電力使用量 (ワット)。
node_gpu_temperature	ClusterName ClusterName , InstanceId , NodeName ClusterName , InstanceId , InstanceType , NodeName, GpuDevice	ノードに割り当てられた GPU の温度 (摂氏)。

メトリクス名	ディメンション	説明
node_gpu_utilization	ClusterName ClusterName , InstanceId , NodeName ClusterName , InstanceId , InstanceType , NodeName, GpuDevice	ノードに割り当てられた GPU の使用率。
pod_gpu_memory_total	ClusterName ClusterName , Namespace ClusterName , Namespace , Service ClusterName , Namespace , PodName ClusterName , Namespace , PodName, FullPodName ClusterName , Namespace , PodName, FullPodName . GpuDevice	ポッドに割り当てられた GPU のフレームバッファの合計サイズ (バイト)。

メトリクス名	ディメンション	説明
pod_gpu_memory_used	ClusterName ClusterName , Namespace ClusterName , Namespace , Service ClusterName , Namespace , PodName ClusterName , Namespace , PodName, FullPodName ClusterName , Namespace , PodName, FullPodName . GpuDevice	ポッドに割り当てられた GPU で使用されているフレームバッファのバイト数。
pod_gpu_memory_utilization	ClusterName ClusterName , Namespace ClusterName , Namespace , Service ClusterName , Namespace , PodName ClusterName , Namespace , PodName, FullPodName ClusterName , Namespace , PodName, FullPodName . GpuDevice	ポッドに割り当てられた GPU で使用されているフレームバッファの割合。

メトリクス名	ディメンション	説明
pod_gpu_power_draw	ClusterName ClusterName , Namespace ClusterName , Namespace , Service ClusterName , Namespace , PodName ClusterName , Namespace , PodName, FullPodName ClusterName , Namespace , PodName, FullPodName . GpuDevice	ポッドに割り当てられた GPU の電力使用量 (ワット)。
pod_gpu_temperature	ClusterName ClusterName , Namespace ClusterName , Namespace , Service ClusterName , Namespace , PodName ClusterName , Namespace , PodName, FullPodName ClusterName , Namespace , PodName, FullPodName . GpuDevice	ポッドに割り当てられた GPU の温度 (摂氏)。

メトリクス名	ディメンション	説明
pod_gpu_utilization	ClusterName ClusterName , Namespace , PodName, ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName , GpuDevice	ポッドに割り当てられた GPU の使用率。

AWS Trainium と AWS Inferentia の AWS Neuron メトリクス

CloudWatch エージェントのバージョン 1.300036.0 以降、Amazon EKS 向けにオブザーバビリティが強化された Container Insights は、デフォルトで AWS Trainium および AWS Inferentia アクセラレーターから高速コンピューティングメトリクスを収集します。CloudWatch エージェントは、CloudWatch Observability EKS アドオンのバージョン v1.5.0-eksbuild.1 以降を使用してインストールする必要があります。アドオンの詳細については、「Amazon CloudWatch Observability EKS アドオンを使用して CloudWatch エージェントをインストールする」を参照してください。AWS Trainium の詳細については、「AWS Trainium」を参照してください。AWS Inferentia の詳細については、「AWS Inferentia」を参照してください。

Container Insights が AWS Neuron メトリクスを収集するには、次の前提条件を満たす必要があります。

- Amazon EKS 向けにオブザーバビリティが強化された Container Insights を使用するには、Amazon CloudWatch Observability EKS アドオンバージョン v1.5.0-eksbuild.1 以降を使用する必要があります。
- Neuron ドライバーはクラスターのノードにインストールする必要があります。
- Neuron デバイスプラグインはクラスターにインストールする必要があります。例えば、Amazon EKS-Optimized Accelerated AMI は、必要なコンポーネントで作成されています。

収集されるメトリクスは、このセクションにある表に一覧表示されています。AWS Trainium、AWS Inferentia、および AWS Inferentia2 のメトリクスが収集されます。

CloudWatch エージェントは Neuron Monitor からこれらのメトリクスを収集し、必要な Kubernetes リソースの関連付けを行って、ポッドレベルとコンテナレベルでメトリクスを配信します。

メトリクス名	ディメンション	説明
container_neuroncore_utilization	ClusterName ClusterName , Namespace , PodName, ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName , NeuronDevice , NeuronCore	コンテナに割り当てられた NeuronCore のキャプチャ期間中の NeuronCore 使用率。 単位: パーセント
container_neuroncore_memory_usage_constants	ClusterName ClusterName , Namespace , PodName, ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName , NeuronDevice , NeuronCore	コンテナに割り当てられた NeuronCore によってトレーニング中に定数 (または推論中の重み) に使用されるデバイスメモリの量。 単位: バイト
container_neuroncore_memory	ClusterName ClusterName , Namespace , PodName, ContainerName	コンテナに割り当てられた NeuronCore によってモデルの実行

メトリクス名	ディメンション	説明
_usage_model_code	ClusterName , Namespace , PodName, FullPodName , ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName , NeuronDevice , NeuronCore	コードに使用されるデバイスメモリの量。 単位: バイト
container_neuroncore_memory_usage_model_shared_scratchpad	ClusterName ClusterName , Namespace , PodName, ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName , NeuronDevice , NeuronCore	コンテナに割り当てられた NeuronCore によってモデルの共有スクラッチパッドに使用されるデバイスメモリの量。このメモリ領域はモデル用に予約されています。 単位: バイト
container_neuroncore_memory_usage_runtime_memory	ClusterName ClusterName , Namespace , PodName, ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName , NeuronDevice , NeuronCore	コンテナに割り当てられた NeuronCore によって Neuron ランタイムに使用されるデバイスメモリの量。 単位: バイト

メトリクス名	ディメンション	説明
container_neuroncore_memory_usage_tensors	ClusterName ClusterName , Namespace , PodName, ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName , NeuronDevice , NeuronCore	コンテナに割り当てられた NeuronCore によってテンソルに使用されるデバイスメモリの量。 単位: バイト
container_neuroncore_memory_usage_total	ClusterName ClusterName , Namespace , PodName, ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName , NeuronDevice , NeuronCore	コンテナに割り当てられた NeuronCore によって使用されるメモリの合計量。 単位: バイト

メトリクス名	ディメンション	説明
container_neurondevice_hw_ecc_events_total	ClusterName ClusterName , Namespace , PodName, ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName , NeuronDevice	ノード上の Neuron デバイスのオンチップ SRAM とデバイスメモリの修正済みおよび未修正 ECC イベントの数。 単位: 個
pod_neuroncore_utilization	ClusterName ClusterName , Namespace ClusterName , Namespace , Service ClusterName , Namespace , PodName, FullPodName ClusterName , Namespace , PodName, FullPodName , NeuronDevice , NeuronCore	ポッドに割り当てられた NeuronCore のキャプチャ期間中の NeuronCore 使用率。 単位: パーセント

メトリクス名	ディメンション	説明
pod_neuroncore_memory_usage_constants	ClusterName ClusterName , Namespace ClusterName , Namespace , Service ClusterName , Namespace , PodName, FullPodName ClusterName , Namespace , PodName, FullPodName , NeuronDevice , NeuronCore	ポッドに割り当てられた NeuronCore によってトレーニング中に定数 (または推論中の重み) に使用されるデバイスメモリの量。 単位: バイト
pod_neuroncore_memory_usage_model_code	ClusterName ClusterName , Namespace ClusterName , Namespace , Service ClusterName , Namespace , PodName, FullPodName ClusterName , Namespace , PodName, FullPodName , NeuronDevice , NeuronCore	ポッドに割り当てられた NeuronCore によってモデルの実行コードに使用されるデバイスメモリの量。 単位: バイト

メトリクス名	ディメンション	説明
pod_neuroncore_memory_usage_model_shared_scratchpad	ClusterName ClusterName , Namespace ClusterName , Namespace , Service ClusterName , Namespace , PodName, FullPodName ClusterName , Namespace , PodName, FullPodName , NeuronDevice , NeuronCore	ポッドに割り当てられた NeuronCore によってモデルの共有スクラッチパッドに使用されるデバイスメモリの量。このメモリ領域はモデル用に予約されています。 単位: バイト
pod_neuroncore_memory_usage_runtime_memory	ClusterName ClusterName , Namespace ClusterName , Namespace , Service ClusterName , Namespace , PodName, FullPodName ClusterName , Namespace , PodName, FullPodName , NeuronDevice , NeuronCore	ポッドに割り当てられた NeuronCore によって Neuron ランタイムに使用されるデバイスメモリの量。 単位: バイト

メトリクス名	ディメンション	説明
pod_neuroncore_memory_usage_tensors	ClusterName ClusterName , Namespace ClusterName , Namespace , Service ClusterName , Namespace , PodName, FullPodName ClusterName , Namespace , PodName, FullPodName , NeuronDevice , NeuronCore	ポッドに割り当てられた NeuronCore によってテンソルに使用されるデバイスメモリの量。 単位: バイト
pod_neuroncore_memory_usage_total	ClusterName ClusterName , Namespace ClusterName , Namespace , Service ClusterName , Namespace , PodName, FullPodName ClusterName , Namespace , PodName, FullPodName , NeuronDevice , NeuronCore	ポッドに割り当てられた NeuronCore によって使用されるメモリの合計量。 単位: バイト

メトリクス名	ディメンション	説明
pod_neurondevice_hw_ecc_events_total	ClusterName ClusterName , Namespace ClusterName , Namespace , Service ClusterName , Namespace , PodName ClusterName , Namespace , PodName, FullPodName ClusterName , Namespace , PodName, FullPodName , NeuronDevice	ポッドに割り当てられた Neuron デバイスのオンチップ SRAM とデバイスメモリの修正済みおよび未修正 ECC イベントの数。 単位: バイト
node_neuroncore_utilization	ClusterName ClusterName , InstanceId , NodeName ClusterName , InstanceType , InstanceId , NodeName, NeuronDevice , NeuronCore	ノードに割り当てられた NeuronCore のキャプチャ期間中の NeuronCore 使用率。 単位: パーセント
node_neuroncore_memory_usage_constants	ClusterName ClusterName , InstanceId , NodeName ClusterName , InstanceType , InstanceId , NodeName, NeuronDevice , NeuronCore	ノードに割り当てられた NeuronCore によってトレーニング中に定数 (または推論中の重み) に使用されるデバイスメモリの量。 単位: バイト

メトリクス名	ディメンション	説明
node_neuroncore_memory_usage_model_code	ClusterName ClusterName , InstanceId , NodeName ClusterName , InstanceType , InstanceId , NodeName, NeuronDevice , NeuronCore	ノードに割り当てられた NeuronCore によってモデルの実行コードに使用されるデバイスメモリの量。 単位: バイト
node_neuroncore_memory_usage_model_shared_scratchpad	ClusterName ClusterName , InstanceId , NodeName ClusterName , InstanceType , InstanceId , NodeName, NeuronDevice , NeuronCore	ノードに割り当てられた NeuronCore によってモデルの共有スクラッチパッドに使用されるデバイスメモリの量。これはモデル用に予約されているメモリ領域です。 単位: バイト
node_neuroncore_memory_usage_runtime_memory	ClusterName ClusterName , InstanceId , NodeName ClusterName , InstanceType , InstanceId , NodeName, NeuronDevice , NeuronCore	ノードに割り当てられた NeuronCore によって Neuron ランタイムに使用されるデバイスメモリの量。 単位: バイト
node_neuroncore_memory_usage_tensors	ClusterName ClusterName , InstanceId , NodeName ClusterName , InstanceType , InstanceId , NodeName, NeuronDevice , NeuronCore	ノードに割り当てられた NeuronCore によってテンソルに使用されるデバイスメモリの量。 単位: バイト

メトリクス名	ディメンション	説明
node_neuroncore_memory_usage_total	ClusterName ClusterName , InstanceId , NodeName ClusterName , InstanceType , InstanceId , NodeName, NeuronDevice , NeuronCore	ノードに割り当てられた NeuronCore によって使用されるメモリの合計量。 単位: バイト
node_neuron_execution_errors_total	ClusterName ClusterName , InstanceId , NodeName	ノード上の実行エラーの合計数。これは、CloudWatch エージェントで generic、numerical 、transient 、model、runtime、hardware の各タイプのエラーを集計することによって計算されます。 単位: 個
node_neurondevice_runtime_memory_used_bytes	ClusterName ClusterName , InstanceId , NodeName	ノード上の Neuron デバイスの合計メモリ使用量 (バイト)。 単位: バイト
node_neuron_execution_latency	ClusterName ClusterName , InstanceId , NodeName	Neuron ランタイムによって測定されたノードでの実行のレイテンシー (秒)。 単位: 秒

メトリクス名	ディメンション	説明
node_neuron device_hw_ecc_events_total	ClusterName ClusterName , InstanceId , NodeName ClusterName , InstanceId , NodeName, NeuronDevice	ノード上の Neuron デバイスのオンチップ SRAM とデバイスメモリの修正済みおよび未修正 ECC イベントの数。 単位: 個

AWS Elastic Fabric Adapter (EFA) メトリクス

CloudWatch エージェントのバージョン 1.300037.0 以降、Amazon EKS 向けにオブザーバビリティが強化された Container Insights は Linux インスタンス上の Amazon EKS クラスターから AWS Elastic Fabric Adapter (EFA) メトリクスを収集します。CloudWatch エージェントは、CloudWatch Observability EKS アドオンのバージョン v1.5.2-eksbuild.1 以降を使用してインストールする必要があります。アドオンの詳細については、「Amazon CloudWatch Observability EKS アドオンを使用して CloudWatch エージェントをインストールする」を参照してください。AWS Elastic Fabric Adapter の詳細については、「Elastic Fabric Adapter」を参照してください。

Container Insights が AWS Elastic Fabric Adapter メトリクスを収集するには、次の前提条件を満たす必要があります。

- Amazon EKS 向けにオブザーバビリティが強化された Container Insights を使用するには、Amazon CloudWatch Observability EKS アドオンバージョン v1.5.2-eksbuild.1 以降を使用する必要があります。
- EFA デバイスプラグインはクラスターにインストールする必要があります。詳細については、GitHub の aws-efa-k8s-device-plugin を参照してください。

収集されるメトリクスのリストを次の表に示します。

メトリクス名	ディメンション	説明
container_efa_rx_bytes	ClusterName ClusterName , Namespace , PodName, ContainerName	コンテナに割り当てられた EFA デバイスによって受信された 1 秒あたりのバイト数。 単位: バイト/秒

メトリクス名	ディメンション	説明
	ClusterName , Namespace , PodName, FullPodName , ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName , EfaDevice	
container_efa_tx_bytes	ClusterName ClusterName , Namespace , PodName, ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName , EfaDevice	コンテナに割り当てられた EFA デバイスによって送信された 1 秒あたりのバイト数。 単位: バイト/秒
container_efa_rx_dropped	ClusterName ClusterName , Namespace , PodName, ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName , EfaDevice	コンテナに割り当てられた EFA デバイスによって受信され、ドロップされたパケットの数。 単位: カウント/秒

メトリクス名	ディメンション	説明
container_efa_rdma_read_bytes	ClusterName ClusterName , Namespace , PodName, ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName , EfaDevice	コンテナに割り当てられた EFA デバイスによって、リモートダイレクトメモリアクセスの読み取りオペレーションを使用して受信された 1 秒あたりのバイト数。 単位: バイト/秒
container_efa_rdma_write_bytes	ClusterName ClusterName , Namespace , PodName, ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName , EfaDevice	コンテナに割り当てられた EFA デバイスによって、リモートダイレクトメモリアクセスの読み取りオペレーションを使用して送信された 1 秒あたりのバイト数。 単位: バイト/秒

メトリクス名	ディメンション	説明
container_efa_rdma_write_recv_bytes	ClusterName ClusterName , Namespace , PodName, ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName , EfaDevice	コンテナに割り当てられた EFA デバイスによって、リモートダイレクトメモリアクセスの書き込みオペレーションを使用して受信された 1 秒あたりのバイト数。 単位: バイト/秒
pod_efa_rx_bytes	ClusterName ClusterName , Namespace ClusterName , Namespace , Service ClusterName , Namespace , PodName ClusterName , Namespace , PodName, FullPodName ClusterName , Namespace , PodName, FullPodName , EfaDevice	ポッドに割り当てられた EFA デバイスによって受信された 1 秒あたりのバイト数。 単位: バイト/秒

メトリクス名	ディメンション	説明
pod_efa_tx_bytes	ClusterName ClusterName , Namespace ClusterName , Namespace , Service ClusterName , Namespace , PodName ClusterName , Namespace , PodName, FullPodName ClusterName , Namespace , PodName, FullPodName , EfaDevice	ポッドに割り当てられた EFA デバイスによって送信された 1 秒あたりのバイト数。 単位: バイト/秒
pod_efa_rx_dropped	ClusterName ClusterName , Namespace ClusterName , Namespace , Service ClusterName , Namespace , PodName ClusterName , Namespace , PodName, FullPodName ClusterName , Namespace , PodName, FullPodName , EfaDevice	ポッドに割り当てられた EFA デバイスによって受信され、ドロップされたパケットの数。 単位: カウント/秒

メトリクス名	ディメンション	説明
pod_efa_rdma_read_bytes	ClusterName ClusterName , Namespace ClusterName , Namespace , Service ClusterName , Namespace , PodName ClusterName , Namespace , PodName, FullPodName ClusterName , Namespace , PodName, FullPodName , EfaDevice	ポッドに割り当てられた EFA デバイスによって、リモートダイレクトメモリアクセスの読み取りオペレーションを使用して受信された 1 秒あたりのバイト数。 単位: バイト/秒
pod_efa_rdma_write_bytes	ClusterName ClusterName , Namespace ClusterName , Namespace , Service ClusterName , Namespace , PodName ClusterName , Namespace , PodName, FullPodName ClusterName , Namespace , PodName, FullPodName , EfaDevice	ポッドに割り当てられた EFA デバイスによって、リモートダイレクトメモリアクセスの読み取りオペレーションを使用して送信された 1 秒あたりのバイト数。 単位: バイト/秒

メトリクス名	ディメンション	説明
pod_efa_rdma_write_recv_bytes	ClusterName ClusterName , Namespace ClusterName , Namespace , Service ClusterName , Namespace , PodName ClusterName , Namespace , PodName, FullPodName ClusterName , Namespace , PodName, FullPodName , EfaDevice	ポッドに割り当てられた EFA デバイスによって、リモートダイレクトメモリアクセスの書き込みオペレーションを使用して受信された 1 秒あたりのバイト数。 単位: バイト/秒
node_efa_rx_bytes	ClusterName ClusterName , InstanceId , NodeName ClusterName , InstanceId , InstanceType , NodeName, EfaDevice	ノードに割り当てられた EFA デバイスによって受信された 1 秒あたりのバイト数。 単位: バイト/秒
node_efa_tx_bytes	ClusterName ClusterName , InstanceId , NodeName ClusterName , InstanceId , InstanceType , NodeName, EfaDevice	ノードに割り当てられた EFA デバイスによって送信された 1 秒あたりのバイト数。 単位: バイト/秒

メトリクス名	ディメンション	説明
node_efa_rx_dropped	ClusterName ClusterName , InstanceId , NodeName ClusterName , InstanceId , InstanceType , NodeName, EfaDevice	ノードに割り当てられた EFA デバイスによって受信され、ドロップされたパケットの数。 単位: カウント/秒
node_efa_rdma_read_bytes	ClusterName ClusterName , InstanceId , NodeName ClusterName , InstanceId , InstanceType , NodeName, EfaDevice	ノードに割り当てられた EFA デバイスによって、リモートダイレクトメモリアクセスの読み取りオペレーションを使用して受信された 1 秒あたりのバイト数。 単位: バイト/秒
pod_efa_rdma_write_bytes	ClusterName ClusterName , InstanceId , NodeName ClusterName , InstanceId , InstanceType , NodeName, EfaDevice	ポッドに割り当てられた EFA デバイスによって、リモートダイレクトメモリアクセスの読み取りオペレーションを使用して送信された 1 秒あたりのバイト数。 単位: バイト/秒
node_efa_rdma_write_recv_bytes	ClusterName ClusterName , InstanceId , NodeName ClusterName , InstanceId , InstanceType , NodeName, EfaDevice	ノードに割り当てられた EFA デバイスによって、リモートダイレクトメモリアクセスの書き込みオペレーションを使用して受信された 1 秒あたりのバイト数。 単位: バイト/秒

Container Insights パフォーマンスログリファレンス

このセクションには、Container Insights がパフォーマンスログイベントを使用してメトリクスを収集する方法に関するリファレンス情報が含まれています。Container Insights をデプロイする場合、パフォーマンスログイベント用のロググループが自動的に作成されます。このロググループを手動で作成する必要はありません。

トピック

Amazon ECS の Container Insights パフォーマンスログイベント

次に、Container Insights が Amazon ECS から収集するパフォーマンスログイベントの例を示します。

これらのログは、CloudWatch Logs の `/aws/ecs/containerinsights/`*`CLUSTER_NAME`*`/performance` という名前のロググループにあります。そのロググループ内では、各コンテナインスタンスに `AgentTelemetry-`*`CONTAINER_INSTANCE_ID`* という名前のログストリームがあります。

すべてのコンテナログイベントを表示する `{ $.Type = "Container" }` などのクエリを使用して、これらのログをクエリできます。

型: コンテナ

```
{
 "Version":"0",
 "Type":"Container",
 "ContainerName":"sleep",
 "TaskId":"7ac4dfba69214411b4783a3b8189c9ba",
 "TaskDefinitionFamily":"sleep360",
 "TaskDefinitionRevision":"1",
 "ContainerInstanceId":"0d7650e6dec34c1a9200f72098071e8f",
 "EC2InstanceId":"i-0c470579dbcdbd2f3",
 "ClusterName":"MyCluster",
 "Image":"busybox",
 "ContainerKnownStatus":"RUNNING",
```

```
 "Timestamp":1623963900000,
 "CpuUtilized":0.0,
 "CpuReserved":10.0,
 "MemoryUtilized":0,
 "MemoryReserved":10,
 "StorageReadBytes":0,
 "StorageWriteBytes":0,
 "NetworkRxBytes":0,
 "NetworkRxDropped":0,
 "NetworkRxErrors":0,
 "NetworkRxPackets":14,
 "NetworkTxBytes":0,
 "NetworkTxDropped":0,
 "NetworkTxErrors":0,
 "NetworkTxPackets":0
}
```

型: タスク

```
{
    "Version": "0",
    "Type": "Task",
    "TaskId": "7ac4dfba69214411b4783a3b8189c9ba",
    "TaskDefinitionFamily": "sleep360",
    "TaskDefinitionRevision": "1",
    "ContainerInstanceId": "0d7650e6dec34c1a9200f72098071e8f",
    "EC2InstanceId": "i-0c470579dbcdbd2f3",
    "ClusterName": "MyCluster",
    "AccountID": "637146863587",
    "Region": "us-west-2",
    "AvailabilityZone": "us-west-2b",
    "KnownStatus": "RUNNING",
    "LaunchType": "EC2",
    "PullStartedAt": 1623963608201,
    "PullStoppedAt": 1623963610065,
    "CreatedAt": 1623963607094,
    "StartedAt": 1623963610382,
    "Timestamp": 1623963900000,
    "CpuUtilized": 0.0,
    "CpuReserved": 10.0,
    "MemoryUtilized": 0,
    "MemoryReserved": 10,
    "StorageReadBytes": 0,
```

```
    "StorageWriteBytes": 0,
    "NetworkRxBytes": 0,
    "NetworkRxDropped": 0,
    "NetworkRxErrors": 0,
    "NetworkRxPackets": 14,
    "NetworkTxBytes": 0,
    "NetworkTxDropped": 0,
    "NetworkTxErrors": 0,
    "NetworkTxPackets": 0,
    "EBSFilesystemUtilized": 10,
    "EBSFilesystemSize": 20,
    "CloudWatchMetrics": [
        {
            "Namespace": "ECS/ContainerInsights",
            "Metrics": [
                {
                    "Name": "CpuUtilized",
                    "Unit": "None"
                },
                {
                    "Name": "CpuReserved",
                    "Unit": "None"
                },
                {
                    "Name": "MemoryUtilized",
                    "Unit": "Megabytes"
                },
                {
                    "Name": "MemoryReserved",
                    "Unit": "Megabytes"
                },
                {
                    "Name": "StorageReadBytes",
                    "Unit": "Bytes/Second"
                },
                {
                    "Name": "StorageWriteBytes",
                    "Unit": "Bytes/Second"
                },
                {
                    "Name": "NetworkRxBytes",
                    "Unit": "Bytes/Second"
                },
                {
```

```
                    "Name": "NetworkTxBytes",
                    "Unit": "Bytes/Second"
                },
                {
                    "Name": "EBSFilesystemSize",
                    "Unit": "Gigabytes"
                },
                {
                    "Name": "EBSFilesystemUtilzed",
                    "Unit": "Gigabytes"
                }
            ],
            "Dimensions": [
                ["ClusterName"],
                [
                    "ClusterName",
                    "TaskDefinitionFamily"
                ]
            ]
        }
    ]
}
```

タイプ: Service

```
{
    "Version": "0",
    "Type": "Service",
    "ServiceName": "myCIService",
    "ClusterName": "myCICluster",
    "Timestamp": 1561586460000,
    "DesiredTaskCount": 2,
    "RunningTaskCount": 2,
    "PendingTaskCount": 0,
    "DeploymentCount": 1,
    "TaskSetCount": 0,
    "CloudWatchMetrics": [
        {
            "Namespace": "ECS/ContainerInsights",
            "Metrics": [
                {
                    "Name": "DesiredTaskCount",
                    "Unit": "Count"
```

```
                },
                {
                    "Name": "RunningTaskCount",
                    "Unit": "Count"
                },
                {
                    "Name": "PendingTaskCount",
                    "Unit": "Count"
                },
                {
                    "Name": "DeploymentCount",
                    "Unit": "Count"
                },
                {
                    "Name": "TaskSetCount",
                    "Unit": "Count"
                }
            ],
            "Dimensions": [
                [
                    "ServiceName",
                    "ClusterName"
                ]
            ]
        }
    ]
}
```

タイプ: ボリューム

```
{
    "Version": "0",
    "Type": "Volume",
    "TaskDefinitionFamily": "myCITaskDef",
    "TaskId": "7ac4dfba69214411b4783a3b8189c9ba",
    "ClusterName": "myCICluster",
    "ServiceName": "myCIService",
    "VolumeId": "vol-1233436545ff708cb",
    "InstanceId": "i-0c470579dbcdbd2f3",
    "LaunchType": "EC2",
    "VolumeName": "MyVolumeName",
    "EBSFilesystemUtilized": 10,
    "EBSFilesystemSize": 20,
```

```
    "CloudWatchMetrics": [
        {
            "Namespace": "ECS/ContainerInsights",
            "Metrics": [
                {
                    "Name": "EBSFilesystemSize",
                    "Unit": "Gigabytes"
                },
                {
                    "Name": "EBSFilesystemUtilzed",
                    "Unit": "Gigabytes"
                }
            ],
            "Dimensions": [
                ["ClusterName"],
                [
                    "VolumeName",
                    "TaskDefinitionFamily",
                    "ClusterName"
                ],
                [
                    "ServiceName",
                    "ClusterName"
                ]
            ]
        }
    ]
}
```

型: クラスター

```
{
    "Version": "0",
    "Type": "Cluster",
    "ClusterName": "myCICluster",
    "Timestamp": 1561587300000,
    "TaskCount": 5,
    "ContainerInstanceCount": 5,
    "ServiceCount": 2,
    "CloudWatchMetrics": [
        {
            "Namespace": "ECS/ContainerInsights",
            "Metrics": [
```

```
                {
                    "Name": "TaskCount",
                    "Unit": "Count"
                },
                {
                    "Name": "ContainerInstanceCount",
                    "Unit": "Count"
                },
                {
                    "Name": "ServiceCount",
                    "Unit": "Count"
                }
            ],
            "Dimensions": [
                [
                    "ClusterName"
                ]
            ]
        }
    ]
}
```

Amazon EKS および Kubernetes の Container Insights パフォーマンスログイベント

次に、Container Insights が Amazon EKS および Kubernetes から収集するパフォーマンスログイベントの例を示します。

タイプ: Node

```
{
  "AutoScalingGroupName": "eksctl-myCICluster-nodegroup-standard-workers-
NodeGroup-1174PV2WHZAYU",
  "CloudWatchMetrics": [
    {
      "Metrics": [
        {
          "Unit": "Percent",
          "Name": "node_cpu_utilization"
        },
        {
          "Unit": "Percent",
          "Name": "node_memory_utilization"
        },
```

```
        {
          "Unit": "Bytes/Second",
          "Name": "node_network_total_bytes"
        },
        {
          "Unit": "Percent",
          "Name": "node_cpu_reserved_capacity"
        },
        {
          "Unit": "Percent",
          "Name": "node_memory_reserved_capacity"
        },
        {
          "Unit": "Count",
          "Name": "node_number_of_running_pods"
        },
        {
          "Unit": "Count",
          "Name": "node_number_of_running_containers"
        }
      ],
      "Dimensions": [
        [
          "NodeName",
          "InstanceId",
          "ClusterName"
        ]
      ],
      "Namespace": "ContainerInsights"
    },
    {
      "Metrics": [
        {
          "Unit": "Percent",
          "Name": "node_cpu_utilization"
        },
        {
          "Unit": "Percent",
          "Name": "node_memory_utilization"
        },
        {
          "Unit": "Bytes/Second",
          "Name": "node_network_total_bytes"
        },
```

```
        {
          "Unit": "Percent",
          "Name": "node_cpu_reserved_capacity"
        },
        {
          "Unit": "Percent",
          "Name": "node_memory_reserved_capacity"
        },
        {
          "Unit": "Count",
          "Name": "node_number_of_running_pods"
        },
        {
          "Unit": "Count",
          "Name": "node_number_of_running_containers"
        },
        {
          "Name": "node_cpu_usage_total"
        },
        {
          "Name": "node_cpu_limit"
        },
        {
          "Unit": "Bytes",
          "Name": "node_memory_working_set"
        },
        {
          "Unit": "Bytes",
          "Name": "node_memory_limit"
        }
      ],
      "Dimensions": [
        [
          "ClusterName"
        ]
      ],
      "Namespace": "ContainerInsights"
    }
  ],
  "ClusterName": "myCICluster",
  "InstanceId": "i-1234567890123456",
  "InstanceType": "t3.xlarge",
  "NodeName": "ip-192-0-2-0.us-west-2.compute.internal",
  "Sources": [
```

```
    "cadvisor",
    "/proc",
    "pod",
    "calculated"
  ],
  "Timestamp": "1567096682364",
  "Type": "Node",
  "Version": "0",
  "kubernetes": {
    "host": "ip-192-168-75-26.us-west-2.compute.internal"
  },
  "node_cpu_limit": 4000,
  "node_cpu_request": 1130,
  "node_cpu_reserved_capacity": 28.249999999999996,
  "node_cpu_usage_system": 33.794636630852764,
  "node_cpu_usage_total": 136.47852169244098,
  "node_cpu_usage_user": 71.67075111567326,
  "node_cpu_utilization": 3.4119630423110245,
  "node_memory_cache": 3103297536,
  "node_memory_failcnt": 0,
  "node_memory_hierarchical_pgfault": 0,
  "node_memory_hierarchical_pgmajfault": 0,
  "node_memory_limit": 16624865280,
  "node_memory_mapped_file": 406646784,
  "node_memory_max_usage": 4230746112,
  "node_memory_pgfault": 0,
  "node_memory_pgmajfault": 0,
  "node_memory_request": 1115684864,
  "node_memory_reserved_capacity": 6.7109407818311055,
  "node_memory_rss": 798146560,
  "node_memory_swap": 0,
  "node_memory_usage": 3901444096,
  "node_memory_utilization": 6.601302600149552,
  "node_memory_working_set": 1097457664,
  "node_network_rx_bytes": 35918.392817386324,
  "node_network_rx_dropped": 0,
  "node_network_rx_errors": 0,
  "node_network_rx_packets": 157.67565245448117,
  "node_network_total_bytes": 68264.20276554905,
  "node_network_tx_bytes": 32345.80994816272,
  "node_network_tx_dropped": 0,
  "node_network_tx_errors": 0,
  "node_network_tx_packets": 154.21455923431654,
  "node_number_of_running_containers": 16,
```

```
  "node_number_of_running_pods": 13
}
```

タイプ: NodeFS

```
{
  "AutoScalingGroupName": "eksctl-myCICluster-nodegroup-standard-workers-
NodeGroup-1174PV2WHZAYU",
  "CloudWatchMetrics": [
    {
      "Metrics": [
        {
          "Unit": "Percent",
          "Name": "node_filesystem_utilization"
        }
      ],
      "Dimensions": [
        [
          "NodeName",
          "InstanceId",
          "ClusterName"
        ],
        [
          "ClusterName"
        ]
      ],
      "Namespace": "ContainerInsights"
    }
  ],
  "ClusterName": "myCICluster",
  "EBSVolumeId": "aws://us-west-2b/vol-0a53108976d4a2fda",
  "InstanceId": "i-1234567890123456",
  "InstanceType": "t3.xlarge",
  "NodeName": "ip-192-0-2-0.us-west-2.compute.internal",
  "Sources": [
    "cadvisor",
    "calculated"
  ],
  "Timestamp": "1567097939726",
  "Type": "NodeFS",
  "Version": "0",
  "device": "/dev/nvme0n1p1",
  "fstype": "vfs",
```

```
  "kubernetes": {
    "host": "ip-192-168-75-26.us-west-2.compute.internal"
  },
  "node_filesystem_available": 17298395136,
  "node_filesystem_capacity": 21462233088,
  "node_filesystem_inodes": 10484720,
  "node_filesystem_inodes_free": 10367158,
  "node_filesystem_usage": 4163837952,
  "node_filesystem_utilization": 19.400767547940255
}
```

タイプ: NodeDiskIO

```
{
  "AutoScalingGroupName": "eksctl-myCICluster-nodegroup-standard-workers-
NodeGroup-1174PV2WHZAYU",
  "ClusterName": "myCICluster",
  "EBSVolumeId": "aws://us-west-2b/vol-0a53108976d4a2fda",
  "InstanceId": "i-1234567890123456",
  "InstanceType": "t3.xlarge",
  "NodeName": "ip-192-0-2-0.us-west-2.compute.internal",
  "Sources": [
    "cadvisor"
  ],
  "Timestamp": "1567096928131",
  "Type": "NodeDiskIO",
  "Version": "0",
  "device": "/dev/nvme0n1",
  "kubernetes": {
    "host": "ip-192-168-75-26.us-west-2.compute.internal"
  },
  "node_diskio_io_service_bytes_async": 9750.505814277016,
  "node_diskio_io_service_bytes_read": 0,
  "node_diskio_io_service_bytes_sync": 230.6174506688036,
  "node_diskio_io_service_bytes_total": 9981.123264945818,
  "node_diskio_io_service_bytes_write": 9981.123264945818,
  "node_diskio_io_serviced_async": 1.153087253344018,
  "node_diskio_io_serviced_read": 0,
  "node_diskio_io_serviced_sync": 0.03603397666700056,
  "node_diskio_io_serviced_total": 1.1891212300110185,
  "node_diskio_io_serviced_write": 1.1891212300110185
}
```

タイプ: NodeNet


```
{
  "AutoScalingGroupName": "eksctl-myCICluster-nodegroup-standard-workers-
NodeGroup-1174PV2WHZAYU",
  "ClusterName": "myCICluster",
  "InstanceId": "i-1234567890123456",
  "InstanceType": "t3.xlarge",
  "NodeName": "ip-192-0-2-0.us-west-2.compute.internal",
  "Sources": [
    "cadvisor",
    "calculated"
  ],
  "Timestamp": "1567096928131",
  "Type": "NodeNet",
  "Version": "0",
  "interface": "eni972f6bfa9a0",
  "kubernetes": {
    "host": "ip-192-168-75-26.us-west-2.compute.internal"
  },
  "node_interface_network_rx_bytes": 3163.008420864309,
  "node_interface_network_rx_dropped": 0,
  "node_interface_network_rx_errors": 0,
  "node_interface_network_rx_packets": 16.575629266820258,
  "node_interface_network_total_bytes": 3518.3935157426017,
  "node_interface_network_tx_bytes": 355.385094878293,
  "node_interface_network_tx_dropped": 0,
  "node_interface_network_tx_errors": 0,
  "node_interface_network_tx_packets": 3.9997714100370625
}
```


タイプ: Pod


```
{
  "AutoScalingGroupName": "eksctl-myCICluster-nodegroup-standard-workers-
NodeGroup-1174PV2WHZAYU",
  "CloudWatchMetrics": [
    {
      "Metrics": [
        {
          "Unit": "Percent",
          "Name": "pod_cpu_utilization"
        },
```

```
        {
          "Unit": "Percent",
          "Name": "pod_memory_utilization"
        },
        {
          "Unit": "Bytes/Second",
          "Name": "pod_network_rx_bytes"
        },
        {
          "Unit": "Bytes/Second",
          "Name": "pod_network_tx_bytes"
        },
        {
          "Unit": "Percent",
          "Name": "pod_cpu_utilization_over_pod_limit"
        },
        {
          "Unit": "Percent",
          "Name": "pod_memory_utilization_over_pod_limit"
        }
      ],
      "Dimensions": [
        [
          "PodName",
          "Namespace",
          "ClusterName"
        ],
        [
          "Service",
          "Namespace",
          "ClusterName"
        ],
        [
          "Namespace",
          "ClusterName"
        ],
        [
          "ClusterName"
        ]
      ],
      "Namespace": "ContainerInsights"
    },
    {
      "Metrics": [
```

```
        {
          "Unit": "Percent",
          "Name": "pod_cpu_reserved_capacity"
        },
        {
          "Unit": "Percent",
          "Name": "pod_memory_reserved_capacity"
        }
      ],
      "Dimensions": [
        [
          "PodName",
          "Namespace",
          "ClusterName"
        ],
        [
          "ClusterName"
        ]
      ],
      "Namespace": "ContainerInsights"
    },
    {
      "Metrics": [
        {
          "Unit": "Count",
          "Name": "pod_number_of_container_restarts"
        }
      ],
      "Dimensions": [
        [
          "PodName",
          "Namespace",
          "ClusterName"
        ]
      ],
      "Namespace": "ContainerInsights"
    }
  ],
  "ClusterName": "myCICluster",
  "InstanceId": "i-1234567890123456",
  "InstanceType": "t3.xlarge",
  "Namespace": "amazon-cloudwatch",
  "NodeName": "ip-192-0-2-0.us-west-2.compute.internal",
  "PodName": "cloudwatch-agent-statsd",
```

```
  "Service": "cloudwatch-agent-statsd",
  "Sources": [
    "cadvisor",
    "pod",
    "calculated"
  ],
  "Timestamp": "1567097351092",
  "Type": "Pod",
  "Version": "0",
  "kubernetes": {
    "host": "ip-192-168-75-26.us-west-2.compute.internal",
    "labels": {
      "app": "cloudwatch-agent-statsd",
      "pod-template-hash": "df44f855f"
    },
    "namespace_name": "amazon-cloudwatch",
    "pod_id": "2f4ff5ac-c813-11e9-a31d-06e9dde32928",
    "pod_name": "cloudwatch-agent-statsd-df44f855f-ts4q2",
    "pod_owners": [
      {
        "owner_kind": "Deployment",
        "owner_name": "cloudwatch-agent-statsd"
      }
    ],
    "service_name": "cloudwatch-agent-statsd"
  },
  "pod_cpu_limit": 200,
  "pod_cpu_request": 200,
  "pod_cpu_reserved_capacity": 5,
  "pod_cpu_usage_system": 1.4504841104992765,
  "pod_cpu_usage_total": 5.817016867430125,
  "pod_cpu_usage_user": 1.1281543081661038,
  "pod_cpu_utilization": 0.14542542168575312,
  "pod_cpu_utilization_over_pod_limit": 2.9085084337150624,
  "pod_memory_cache": 8192,
  "pod_memory_failcnt": 0,
  "pod_memory_hierarchical_pgfault": 0,
  "pod_memory_hierarchical_pgmajfault": 0,
  "pod_memory_limit": 104857600,
  "pod_memory_mapped_file": 0,
  "pod_memory_max_usage": 25268224,
  "pod_memory_pgfault": 0,
  "pod_memory_pgmajfault": 0,
  "pod_memory_request": 104857600,
```

```
  "pod_memory_reserved_capacity": 0.6307275170893897,
  "pod_memory_rss": 22777856,
  "pod_memory_swap": 0,
  "pod_memory_usage": 25141248,
  "pod_memory_utilization": 0.10988455961791709,
  "pod_memory_utilization_over_pod_limit": 17.421875,
  "pod_memory_working_set": 18268160,
  "pod_network_rx_bytes": 9880.697124714186,
  "pod_network_rx_dropped": 0,
  "pod_network_rx_errors": 0,
  "pod_network_rx_packets": 107.80005532263283,
  "pod_network_total_bytes": 10158.829201483635,
  "pod_network_tx_bytes": 278.13207676944796,
  "pod_network_tx_dropped": 0,
  "pod_network_tx_errors": 0,
  "pod_network_tx_packets": 1.146027574644318,
  "pod_number_of_container_restarts": 0,
  "pod_number_of_containers": 1,
  "pod_number_of_running_containers": 1,
  "pod_status": "Running"
}
```


タイプ: PodNet


```
{
  "AutoScalingGroupName": "eksctl-myCICluster-nodegroup-standard-workers-
NodeGroup-1174PV2WHZAYU",
  "ClusterName": "myCICluster",
  "InstanceId": "i-1234567890123456",
  "InstanceType": "t3.xlarge",
  "Namespace": "amazon-cloudwatch",
  "NodeName": "ip-192-0-2-0.us-west-2.compute.internal",
  "PodName": "cloudwatch-agent-statsd",
  "Service": "cloudwatch-agent-statsd",
  "Sources": [
    "cadvisor",
    "calculated"
  ],
  "Timestamp": "1567097351092",
  "Type": "PodNet",
  "Version": "0",
  "interface": "eth0",
  "kubernetes": {
```

```
    "host": "ip-192-168-75-26.us-west-2.compute.internal",
    "labels": {
      "app": "cloudwatch-agent-statsd",
      "pod-template-hash": "df44f855f"
    },
    "namespace_name": "amazon-cloudwatch",
    "pod_id": "2f4ff5ac-c813-11e9-a31d-06e9dde32928",
    "pod_name": "cloudwatch-agent-statsd-df44f855f-ts4q2",
    "pod_owners": [
      {
        "owner_kind": "Deployment",
        "owner_name": "cloudwatch-agent-statsd"
      }
    ],
    "service_name": "cloudwatch-agent-statsd"
  },
  "pod_interface_network_rx_bytes": 9880.697124714186,
  "pod_interface_network_rx_dropped": 0,
  "pod_interface_network_rx_errors": 0,
  "pod_interface_network_rx_packets": 107.80005532263283,
  "pod_interface_network_total_bytes": 10158.829201483635,
  "pod_interface_network_tx_bytes": 278.13207676944796,
  "pod_interface_network_tx_dropped": 0,
  "pod_interface_network_tx_errors": 0,
  "pod_interface_network_tx_packets": 1.146027574644318
}
```

型: コンテナ

```
{
  "AutoScalingGroupName": "eksctl-myCICluster-nodegroup-standard-workers-NodeGroup-
sample",
  "ClusterName": "myCICluster",
  "InstanceId": "i-1234567890123456",
  "InstanceType": "t3.xlarge",
  "Namespace": "amazon-cloudwatch",
  "NodeName": "ip-192-0-2-0.us-west-2.compute.internal",
  "PodName": "cloudwatch-agent-statsd",
  "Service": "cloudwatch-agent-statsd",
  "Sources": [
    "cadvisor",
    "pod",
    "calculated"
```

```
  ],
  "Timestamp": "1567097399912",
  "Type": "Container",
  "Version": "0",
  "container_cpu_limit": 200,
  "container_cpu_request": 200,
  "container_cpu_usage_system": 1.87958283771964,
  "container_cpu_usage_total": 6.159993652997942,
  "container_cpu_usage_user": 1.6707403001952357,
  "container_cpu_utilization": 0.15399984132494854,
  "container_memory_cache": 8192,
  "container_memory_failcnt": 0,
  "container_memory_hierarchical_pgfault": 0,
  "container_memory_hierarchical_pgmajfault": 0,
  "container_memory_limit": 104857600,
  "container_memory_mapped_file": 0,
  "container_memory_max_usage": 24580096,
  "container_memory_pgfault": 0,
  "container_memory_pgmajfault": 0,
  "container_memory_request": 104857600,
  "container_memory_rss": 22736896,
  "container_memory_swap": 0,
  "container_memory_usage": 24453120,
  "container_memory_utilization": 0.10574541028701798,
  "container_memory_working_set": 17580032,
  "container_status": "Running",
  "kubernetes": {
    "container_name": "cloudwatch-agent",
    "docker": {
      "container_id":
"8967b6b37da239dfad197c9fdea3e5dfd35a8a759ec86e2e4c3f7b401e232706"
    },
    "host": "ip-192-168-75-26.us-west-2.compute.internal",
    "labels": {
      "app": "cloudwatch-agent-statsd",
      "pod-template-hash": "df44f855f"
    },
    "namespace_name": "amazon-cloudwatch",
    "pod_id": "2f4ff5ac-c813-11e9-a31d-06e9dde32928",
    "pod_name": "cloudwatch-agent-statsd-df44f855f-ts4q2",
    "pod_owners": [
      {
        "owner_kind": "Deployment",
        "owner_name": "cloudwatch-agent-statsd"
```

```
      }
    ],
    "service_name": "cloudwatch-agent-statsd"
  },
  "number_of_container_restarts": 0
}
```

タイプ: ContainerFS

```
{
  "AutoScalingGroupName": "eksctl-myCICluster-nodegroup-standard-workers-
NodeGroup-1174PV2WHZAYU",
  "ClusterName": "myCICluster",
  "EBSVolumeId": "aws://us-west-2b/vol-0a53108976d4a2fda",
  "InstanceId": "i-1234567890123456",
  "InstanceType": "t3.xlarge",
  "Namespace": "amazon-cloudwatch",
  "NodeName": "ip-192-0-2-0.us-west-2.compute.internal",
  "PodName": "cloudwatch-agent-statsd",
  "Service": "cloudwatch-agent-statsd",
  "Sources": [
    "cadvisor",
    "calculated"
  ],
  "Timestamp": "1567097399912",
  "Type": "ContainerFS",
  "Version": "0",

  "device": "/dev/nvme0n1p1",
  "fstype": "vfs",
  "kubernetes": {
    "container_name": "cloudwatch-agent",
    "docker": {
      "container_id":
 "8967b6b37da239dfad197c9fdea3e5dfd35a8a759ec86e2e4c3f7b401e232706"
    },
    "host": "ip-192-168-75-26.us-west-2.compute.internal",
    "labels": {
      "app": "cloudwatch-agent-statsd",
      "pod-template-hash": "df44f855f"
    },
    "namespace_name": "amazon-cloudwatch",
    "pod_id": "2f4ff5ac-c813-11e9-a31d-06e9dde32928",
```

```
    "pod_name": "cloudwatch-agent-statsd-df44f855f-ts4q2",
    "pod_owners": [
      {
        "owner_kind": "Deployment",
        "owner_name": "cloudwatch-agent-statsd"
      }
    ],
    "service_name": "cloudwatch-agent-statsd"
  }
}
```

型: クラスター

```
{
  "CloudWatchMetrics": [
    {
      "Metrics": [
        {
          "Unit": "Count",
          "Name": "cluster_node_count"
        },
        {
          "Unit": "Count",
          "Name": "cluster_failed_node_count"
        }
      ],
      "Dimensions": [
        [
          "ClusterName"
        ]
      ],
      "Namespace": "ContainerInsights"
    }
  ],
  "ClusterName": "myCICluster",
  "Sources": [
    "apiserver"
  ],
  "Timestamp": "1567097534160",
  "Type": "Cluster",
  "Version": "0",
  "cluster_failed_node_count": 0,
  "cluster_node_count": 3
```

```
}
```

タイプ: ClusterService

```
{
  "CloudWatchMetrics": [
    {
      "Metrics": [
        {
          "Unit": "Count",
          "Name": "service_number_of_running_pods"
        }
      ],
      "Dimensions": [
        [
          "Service",
          "Namespace",
          "ClusterName"
        ],
        [
          "ClusterName"
        ]
      ],
      "Namespace": "ContainerInsights"
    }
  ],
  "ClusterName": "myCICluster",
  "Namespace": "amazon-cloudwatch",
  "Service": "cloudwatch-agent-statsd",
  "Sources": [
    "apiserver"
  ],
  "Timestamp": "1567097534160",
  "Type": "ClusterService",
  "Version": "0",
  "kubernetes": {
    "namespace_name": "amazon-cloudwatch",
    "service_name": "cloudwatch-agent-statsd"
  },
  "service_number_of_running_pods": 1
}
```

タイプ: ClusterNamespace

```
{
  "CloudWatchMetrics": [
    {
      "Metrics": [
        {
          "Unit": "Count",
          "Name": "namespace_number_of_running_pods"
        }
      ],
      "Dimensions": [
        [
          "Namespace",
          "ClusterName"
        ],
        [
          "ClusterName"
        ]
      ],
      "Namespace": "ContainerInsights"
    }
  ],
  "ClusterName": "myCICluster",
  "Namespace": "amazon-cloudwatch",
  "Sources": [
    "apiserver"
  ],
  "Timestamp": "1567097594160",
  "Type": "ClusterNamespace",
  "Version": "0",
  "kubernetes": {
    "namespace_name": "amazon-cloudwatch"
  },
  "namespace_number_of_running_pods": 7
}
```


Amazon EKS と Kubernetes のパフォーマンスログイベントの関連フィールド

Amazon EKS および Kubernetes では、コンテナ化された CloudWatch エージェントは、パフォーマンスログイベントとしてデータを出力します。これにより、CloudWatch は高カーディナリティのデータを取り込み、保存することができます。CloudWatch は、パフォーマンスログイベントのデータを使用して、詳細を失うことなく、クラスター、ノード、ポッドレベルで、集計された CloudWatch メトリクスを作成します。

以下の表は、Container Insights メトリクスデータの収集に関連する、これらのパフォーマンスログイベントのフィールドを示しています。CloudWatch Logs Insights を使用して、これらの任意のフィールドをクエリし、データを収集したり、問題を調査したりできます。詳細については、「CloudWatch Logs Insights を使用したログデータの分析」を参照してください。

タイプ	ログフィールド	送信元	計算式またはコメント
ポッド	`pod_cpu_utilization`	計算	計算式: `pod_cpu_usage_total / node_cpu_limit`
ポッド	`pod_cpu_usage_total` `pod_cpu_usage_total` はミリコア単位で報告されます。	cadvisor	
ポッド	`pod_cpu_limit`	計算	計算式: `sum(container_cpu_limit)` `sum(container_cpu_limit)` には、既に完成したポッドが含まれます。 ポッドのいずれかのコンテナで、定義された CPU 制限がない場合、このフィール

タイプ	ログフィールド	送信元	計算式またはコメント
			ドはログイベントに表示されません。これには初期化コンテナが含まれます。
ポッド	pod_cpu_request	計算	計算式: sum(container_cpu_request) container_cpu_request の設定は保証されません。設定されているもののみが合計に含まれます。
ポッド	pod_cpu_utilization_over_pod_limit	計算	計算式: pod_cpu_usage_total / pod_cpu_limit
ポッド	pod_cpu_reserved_capacity	計算	計算式: pod_cpu_request / node_cpu_limit

タイプ	ログフィールド	送信元	計算式またはコメント
ポッド	pod_memory_utilization	計算	計算式: pod_memory_working_set / node_memory_limit これは、ノードのメモリ制限に対する Pod メモリ使用量の割合です。
ポッド	pod_memory_working_set	cadvisor	
ポッド	pod_memory_limit	計算	計算式: sum(container_memory_limit) ポッドのいずれかのコンテナに、定義されたメモリ制限がない場合、このフィールドはログイベントに表示されません。これには初期化コンテナが含まれます。

タイプ	ログフィールド	送信元	計算式またはコメント
ポッド	pod_memory_request	計算	計算式: sum(container_memory_request) container_memory_request の設定は保証されません。設定されているもののみが合計に含まれます。

タイプ	ログフィールド	送信元	計算式またはコメント
ポッド	pod_memory_utilization_over_pod_limit	計算	計算式: pod_memory_working_set / pod_memory_limit ポッドのいずれかのコンテナに、定義されたメモリ制限がない場合、このフィールドはログイベントに表示されません。これには初期化コンテナが含まれます。
ポッド	pod_memory_reserved_capacity	計算	計算式: pod_memory_request / node_memory_limit

タイプ	ログフィールド	送信元	計算式またはコメント
ポッド	pod_network_tx_bytes	計算	計算式: sum(pod_interface_network_tx_bytes) このデータは、ポッドごとにすべてのネットワークインターフェイスで使用できます。CloudWatch エージェントが合計を計算し、メトリクス抽出ルールを追加します。
ポッド	pod_network_rx_bytes	計算	計算式: sum(pod_interface_network_rx_bytes)
ポッド	pod_network_total_bytes	計算	計算式: pod_network_rx_bytes + pod_network_tx_bytes

タイプ	ログフィールド	送信元	計算式またはコメント
PodNet	pod_interface_network_rx_bytes	cadvisor	このデータは、ポッドネットワークインターフェイスの 1 秒あたりのネットワーク rx バイトです。
PodNet	pod_interface_network_tx_bytes	cadvisor	このデータは、ポッドネットワークインターフェイスの 1 秒あたりのネットワーク tx バイトです。
コンテナ	container_cpu_usage_total	cadvisor	
コンテナ	container_cpu_limit	cadvisor	設定は保証されていません。設定されていない場合は出力されません。
コンテナ	container_cpu_request	cadvisor	設定は保証されていません。設定されていない場合は出力されません。
コンテナ	container_memory_working_set	cadvisor	

タイプ	ログフィールド	送信元	計算式またはコメント
コンテナ	container_memory_limit	ポッド	設定は保証されていません。設定されていない場合は出力されません。
コンテナ	container_memory_request	ポッド	設定は保証されていません。設定されていない場合は出力されません。
ノード	node_cpu_utilization	計算	計算式: node_cpu_usage_total / node_cpu_limit
ノード	node_cpu_usage_total	cadvisor	
ノード	node_cpu_limit	/proc	

タイプ	ログフィールド	送信元	計算式またはコメント
ノード	node_cpu_request	計算	計算式: sum(pod_cpu_request) CronJob の場合、node_cpu_request には完了したポッドからのリクエストも含まれます。これにより、node_cpu_reserved_capacity の値が高くなる可能性があります。
ノード	node_cpu_reserved_capacity	計算	計算式: node_cpu_request / node_cpu_limit
ノード	node_memory_utilization	計算	計算式: node_memory_working_set / node_memory_limit
ノード	node_memory_working_set	cadvisor	

タイプ	ログフィールド	送信元	計算式またはコメント
ノード	node_memory_limit	/proc	
ノード	node_memory_request	計算	計算式: sum(pod_memory_request)
ノード	node_memory_reserved_capacity	計算	計算式: node_memory_request / node_memory_limit
ノード	node_network_rx_bytes	計算	計算式: sum(node_interface_network_rx_bytes)
ノード	node_network_tx_bytes	計算	計算式: sum(node_interface_network_tx_bytes)
ノード	node_network_total_bytes	計算	計算式: node_network_rx_bytes + node_network_tx_bytes
ノード	node_number_of_running_pods	ポッドリスト	

タイプ	ログフィールド	送信元	計算式またはコメント
ノード	node_number_of_running_containers	ポッドリスト	
NodeNet	node_interface_network_rx_bytes	cadvisor	このデータは、ワーカーノードネットワークインターフェイスの 1 秒あたりのネットワーク rx バイトです。
NodeNet	node_interface_network_tx_bytes	cadvisor	このデータは、ワーカーノードネットワークインターフェイスの 1 秒あたりのネットワーク tx バイトです。
NodeFS	node_filesystem_capacity	cadvisor	
NodeFS	node_filesystem_usage	cadvisor	

タイプ	ログフィールド	送信元	計算式またはコメント
NodeFS	node_filesystem_utilization	計算	計算式: node_filesystem_usage / node_filesystem_capacity このデータはデバイス名ごとに使用できます。
クラスター	cluster_failed_node_count	API サーバー	
クラスター	cluster_node_count	API サーバー	
サービス	service_number_of_running_pods	API サーバー	
Namespace	namespace_number_of_running_pods	API サーバー	

メトリクスの計算例

このセクションには、前述の表の一部の値を計算する方法を示す例が含まれています。

次の状態のクラスターがあるとします。

```
Node1
   node_cpu_limit = 4
   node_cpu_usage_total = 3

   Pod1
     pod_cpu_usage_total = 2

     Container1
```

```
        container_cpu_limit = 1
        container_cpu_request = 1
        container_cpu_usage_total = 0.8

     Container2
        container_cpu_limit = null
        container_cpu_request = null
        container_cpu_usage_total = 1.2

   Pod2
     pod_cpu_usage_total = 0.4

     Container3
        container_cpu_limit = 1
        container_cpu_request = 0.5
        container_cpu_usage_total = 0.4

Node2
   node_cpu_limit = 8
   node_cpu_usage_total = 1.5

   Pod3
     pod_cpu_usage_total = 1

     Container4
        container_cpu_limit = 2
        container_cpu_request = 2
        container_cpu_usage_total = 1
```

以下の表は、このデータを使用してポッド CPU メトリクスがどのように計算されるかを示しています。

メトリクス	計算式	Pod1	Pod2	Pod3
pod_cpu_utilization	pod_cpu_usage_total / node_cpu_limit	2 / 4 = 50%	0.4 / 4 = 10%	1 / 8 = 12.5%
pod_cpu_utilization_over_pod_limit	pod_cpu_usage_total / sum(container_cpu_limit)	該当なし (Container2 の CPU 制限が定義さ	0.4 / 1 = 40%	1 / 2 = 50%

メトリクス	計算式	Pod1	Pod2	Pod3
		れていないため)		
pod_cpu_reserved_capacity	sum(container_cpu_request) / node_cpu_limit	(1 + 0) / 4 = 25%	0.5 / 4 = 12.5%	2 / 8 = 25%

以下の表は、このデータを使用してノードの CPU メトリクスがどのように計算されるかを示しています。

メトリクス	計算式	Node1	Node2
node_cpu_utilization	node_cpu_usage_total / node_cpu_limit	3 / 4 = 75%	1.5 / 8 = 18.75%
node_cpu_reserved_capacity	sum(pod_cpu_request) / node_cpu_limit	1.5 / 4 = 37.5%	2 / 8 = 25%

Container Insights の Prometheus メトリクスのモニターリング

CloudWatch Container Insights で Prometheus をモニターリングすると、コンテナ化されたシステムとワークロードからの Prometheus メトリクスの検出が自動化されます。Prometheus はオープンソースのシステムモニターリングおよび警告ツールキットです。詳細については、Prometheus のドキュメントの「What is Prometheus?」を参照してください。

Prometheus メトリクスの検出は、Amazon Elastic Container Service、Amazon Elastic Kubernetes Service、および Amazon EC2 インスタンスで実行されている Kubernetes クラスターでサポートされています。Prometheus カウンタ、ゲージ、およびサマリーメトリクスタイプが収集されます。ヒストグラムメトリクスのサポートは、今後のリリースで予定されています。

Amazon ECS クラスターと Amazon EKS クラスターでは、EC2 と Fargate の両方の起動タイプがサポートされています。Container Insights は、複数のワークロードからメトリクスを自動的に収集し、任意のワークロードからメトリクスを収集するように設定できます。

Prometheus は、CloudWatch でカスタムメトリクスを取り込むオープンソースかつオープンスタンダードな方法として採用できます。Prometheus をサポートする CloudWatch エージェント

は、Prometheus メトリクスを検出して収集し、アプリケーションのパフォーマンスの低下や障害をより迅速にモニターリング、トラブルシューティング、警告します。これにより、オブザーバビリティを強化するために必要なモニターリングツールの数も削減されます。

Container Insights の Prometheus サポートでは、収集、保存、分析などのメトリクスとログの使用量に応じて課金されます。詳細については、Amazon CloudWatch 料金表をご覧ください。

一部のワークロード用に構築済みのダッシュボード

Container Insights Prometheus ソリューションには、このセクションに記載されている一般的なワークロード用の事前構築済みのダッシュボードが含まれています。これらのワークロードの設定例については、「(オプション) Prometheus メトリクスのテストのためにコンテナ化された Amazon ECS のサンプルワークロードを設定する」および「(オプション) Prometheus メトリクスのテストのためにコンテナ化された Amazon EKS サンプルワークロードを設定する」を参照してください。

エージェント設定ファイルを編集して、他のコンテナ化されたサービスおよびアプリケーションから Prometheus メトリクスを収集するように Container Insights を設定することもできます。

Amazon EC2 インスタンスで実行されている Amazon EKS クラスターおよび Kubernetes クラスター用の事前構築済みのダッシュボードを使用したワークロードには、以下があります。

- AWS App Mesh
- NGINX
- Memcached
- Java/JMX
- HAProxy

Amazon ECS クラスター用の事前構築済みダッシュボードを使用したワークロードには、以下があります。

- AWS App Mesh
- Java/JMX
- NGINX
- NGINX Plus

Amazon ECS クラスターで Prometheus メトリクスコレクションをセットアップおよび設定する

Amazon ECS クラスターから Prometheus メトリクスを収集するには、CloudWatch エージェントをコレクターとして使用するか、AWS Distro for OpenTelemetry コレクターを使用できます。AWS Distro for OpenTelemetry コレクターの使用については、https://aws-otel.github.io/docs/getting-started/container-insights/ecs-prometheus を参照してください。

以下のセクションでは、CloudWatch エージェントをコレクターとして使用して Prometheus メトリクスを取得する方法について説明します。Amazon ECS を実行しているクラスターに Prometheus モニターリングを使用して CloudWatch エージェントをインストールし、オプションで追加のターゲットをスクレイプするようにエージェントを設定することができます。また、これらのセクションでは、Prometheus モニターリングでのテストに使用するサンプルワークロードを設定するためのオプションのチュートリアルも提供します。

Amazon ECS の Container Insights は、Prometheus メトリクスの次の起動タイプとネットワークモードの組み合わせをサポートしています。

Amazon ECS 起動タイプ	サポートされているネットワークモード
EC2 (Linux)	ブリッジ、ホストおよび awsvpc
Fargate	awsvpc

VPC セキュリティグループの要件

Prometheus ワークロードのセキュリティグループの受信ルールでは、Prometheus のメトリクスをプライベート IP でスクレイピングするために、CloudWatch エージェントへの Prometheus ポートを開く必要があります。

CloudWatch エージェントのセキュリティグループの出力ルールでは、CloudWatch エージェントがプライベート IP によって Prometheus ワークロードのポートに接続できるようにする必要があります。

トピック

- Amazon ECS クラスターに Prometheus メトリクスコレクションを持つ CloudWatch エージェントをインストールする
- 追加の Prometheus ソースのスクレイピングと、それらのメトリクスのインポート

- (オプション) Prometheus メトリクスのテストのためにコンテナ化された Amazon ECS のサンプルワークロードを設定する

Amazon ECS クラスターに Prometheus メトリクスコレクションを持つ CloudWatch エージェントをインストールする

このセクションでは、Amazon ECS を実行しているクラスターで Prometheus モニターリングを使用して CloudWatch エージェントをセットアップする方法について説明します。これを行うと、エージェントは、そのクラスターで実行されている次のワークロードのメトリクスを自動的にスクレイプし、インポートします。

- AWS App Mesh
- Java/JMX

また、追加の Prometheus ワークロードとソースからメトリクスをスクレイプしてインポートするようにエージェントを設定することもできます。

IAM ロールの設定

CloudWatch エージェントタスク定義には、2 つの IAM ロールが必要です。Container Insights がこれらのロールを作成するように **`CreateIAMRoles=True`** スタックで AWS CloudFormation を指定すると、ロールは正しいアクセス許可によって作成されます。お客様自身でロールを作成する場合、または既存のロールを使用する場合は、次のロールとアクセス許可が必要です。

- CloudWatch エージェントの ECS タスクロール – CloudWatch エージェントコンテナはこのロールを使用します。これには、CloudWatchAgentServerPolicy ポリシーと、次の読み取り専用のアクセス許可を含むカスタマー管理ポリシーが含まれている必要があります。
  - `ec2:DescribeInstances`
  - `ecs:ListTasks`
  - `ecs:ListServices`
  - `ecs:DescribeContainerInstances`
  - `ecs:DescribeServices`
  - `ecs:DescribeTasks`
  - `ecs:DescribeTaskDefinition`
- CloudWatch エージェント ECS タスク実行ロール – これは、コンテナを起動して実行するために Amazon ECS が必要とするロールです。タスク実行ロールに [AmazonSSMReadOnlyAccess]、

[AmazonECSTaskExecutionRolePolicy]、および [CloudWatchAgentServerPolicy] ポリシーがアタッチされていることを確認します。Amazon ECS で使用するためにより機密性の高いデータを保存する場合は、「機密データの指定」を参照してください。

を使用して、Prometheus モニターリングを使用する CloudWatch エージェントをインストールする AWS CloudFormation

AWS CloudFormation を使用して、Amazon ECS クラスターの Prometheus モニターリングで CloudWatch エージェントをインストールします。次のリストは、AWS CloudFormation テンプレートで使用するパラメータを示しています。

- EcsClusterName – ターゲット Amazon ECS クラスターを指定します。
- CreateIamRoles – Amazon ECS タスク実行ロールと Amazon ECS タスクロールの新しいロールを作成するときに **`True`** を指定します 。既存のロールを再利用する場合に **`False`** を指定します。
- TaskRoleName – [CreateIAMRoles] に **`True`** を指定した場合、新しい Amazon ECS タスクロールに使用する名前を指定します。[CreateIAMRoles] に **`False`** を指定した場合、Amazon ECS タスクロールとして使用する既存のロールを指定します。
- ExecutionRoleName – [CreateIAMRoles] に **`True`** を指定した場合、新しい Amazon ECS タスク実行ロールに使用する名前を指定します。[CreateIAMRoles] に **`False`** を指定した場合、Amazon ECS タスク実行ロールとして使用する既存のロールを指定します。
- ECSNetworkMode – EC2 起動タイプを使用している場合、ここでネットワークモードを指定します。これは **`bridge`** または **`host`** である必要があります。
- ECSLaunchType – **`fargate`** または **`EC2`** を指定します。
- SecurityGroupID – [ECSNetworkMode] が **`awsvpc`** である場合、ここでセキュリティグループ ID を指定します。
- SubnetID – ECSNetworkMode が **`awsvpc`** の場合、ここでサブネット ID を指定します。

コマンドサンプル

このセクションでは、さまざまなシナリオで Prometheus モニターリングを使用して Container Insights をインストールするためのサンプル AWS CloudFormation コマンドについて説明します。

ブリッジネットワークモードで Amazon ECS クラスターの AWS CloudFormation スタックを作成する

```
export AWS_PROFILE=your_aws_config_profile_eg_default
```

```
export AWS_DEFAULT_REGION=your_aws_region_eg_ap-southeast-1
export ECS_CLUSTER_NAME=your_ec2_ecs_cluster_name
export ECS_NETWORK_MODE=bridge
export CREATE_IAM_ROLES=True
export ECS_TASK_ROLE_NAME=your_selected_ecs_task_role_name
export ECS_EXECUTION_ROLE_NAME=your_selected_ecs_execution_role_name

curl -O https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-
insights/latest/ecs-task-definition-templates/deployment-mode/replica-service/cwagent-
prometheus/cloudformation-quickstart/cwagent-ecs-prometheus-metric-for-bridge-host.yaml

aws cloudformation create-stack --stack-name CWAgent-Prometheus-ECS-
${ECS_CLUSTER_NAME}-EC2-${ECS_NETWORK_MODE} \
    --template-body file://cwagent-ecs-prometheus-metric-for-bridge-host.yaml \
    --parameters ParameterKey=ECSClusterName,ParameterValue=${ECS_CLUSTER_NAME} \
                 ParameterKey=CreateIAMRoles,ParameterValue=${CREATE_IAM_ROLES} \
                 ParameterKey=ECSNetworkMode,ParameterValue=${ECS_NETWORK_MODE} \
                 ParameterKey=TaskRoleName,ParameterValue=${ECS_TASK_ROLE_NAME} \
                 ParameterKey=ExecutionRoleName,ParameterValue=
${ECS_EXECUTION_ROLE_NAME} \
    --capabilities CAPABILITY_NAMED_IAM \
    --region ${AWS_DEFAULT_REGION} \
    --profile ${AWS_PROFILE}
```

ホストネットワークモードで Amazon ECS クラスターの AWS CloudFormation スタックを作成する

```
export AWS_PROFILE=your_aws_config_profile_eg_default
export AWS_DEFAULT_REGION=your_aws_region_eg_ap-southeast-1
export ECS_CLUSTER_NAME=your_ec2_ecs_cluster_name
export ECS_NETWORK_MODE=host
export CREATE_IAM_ROLES=True
export ECS_TASK_ROLE_NAME=your_selected_ecs_task_role_name
export ECS_EXECUTION_ROLE_NAME=your_selected_ecs_execution_role_name


curl -O https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-
insights/latest/ecs-task-definition-templates/deployment-mode/replica-service/cwagent-
prometheus/cloudformation-quickstart/cwagent-ecs-prometheus-metric-for-bridge-host.yaml

aws cloudformation create-stack --stack-name CWAgent-Prometheus-ECS-
${ECS_CLUSTER_NAME}-EC2-${ECS_NETWORK_MODE} \
    --template-body file://cwagent-ecs-prometheus-metric-for-bridge-host.yaml \
    --parameters ParameterKey=ECSClusterName,ParameterValue=${ECS_CLUSTER_NAME} \
```

```
                ParameterKey=CreateIAMRoles,ParameterValue=${CREATE_IAM_ROLES} \
                ParameterKey=ECSNetworkMode,ParameterValue=${ECS_NETWORK_MODE} \
                ParameterKey=TaskRoleName,ParameterValue=${ECS_TASK_ROLE_NAME} \
                ParameterKey=ExecutionRoleName,ParameterValue=
${ECS_EXECUTION_ROLE_NAME} \
    --capabilities CAPABILITY_NAMED_IAM \
    --region ${AWS_DEFAULT_REGION} \
    --profile ${AWS_PROFILE}
```

awsvpc ネットワークモードで Amazon ECS クラスターの AWS CloudFormation スタックを作成する

```
export AWS_PROFILE=your_aws_config_profile_eg_default
export AWS_DEFAULT_REGION=your_aws_region_eg_ap-southeast-1
export ECS_CLUSTER_NAME=your_ec2_ecs_cluster_name
export ECS_LAUNCH_TYPE=EC2
export CREATE_IAM_ROLES=True
export ECS_CLUSTER_SECURITY_GROUP=your_security_group_eg_sg-xxxxxxxxxx
export ECS_CLUSTER_SUBNET=your_subnet_eg_subnet-xxxxxxxxxx
export ECS_TASK_ROLE_NAME=your_selected_ecs_task_role_name
export ECS_EXECUTION_ROLE_NAME=your_selected_ecs_execution_role_name

curl -O https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-
insights/latest/ecs-task-definition-templates/deployment-mode/replica-service/cwagent-
prometheus/cloudformation-quickstart/cwagent-ecs-prometheus-metric-for-awsvpc.yaml

aws cloudformation create-stack --stack-name CWAgent-Prometheus-ECS-
${ECS_CLUSTER_NAME}-${ECS_LAUNCH_TYPE}-awsvpc \
    --template-body file://cwagent-ecs-prometheus-metric-for-awsvpc.yaml \
    --parameters ParameterKey=ECSClusterName,ParameterValue=${ECS_CLUSTER_NAME} \
                 ParameterKey=CreateIAMRoles,ParameterValue=${CREATE_IAM_ROLES} \
                 ParameterKey=ECSLaunchType,ParameterValue=${ECS_LAUNCH_TYPE} \
                 ParameterKey=SecurityGroupID,ParameterValue=
${ECS_CLUSTER_SECURITY_GROUP} \
                 ParameterKey=SubnetID,ParameterValue=${ECS_CLUSTER_SUBNET} \
                 ParameterKey=TaskRoleName,ParameterValue=${ECS_TASK_ROLE_NAME} \
                 ParameterKey=ExecutionRoleName,ParameterValue=
${ECS_EXECUTION_ROLE_NAME} \
    --capabilities CAPABILITY_NAMED_IAM \
    --region ${AWS_DEFAULT_REGION} \
    --profile ${AWS_PROFILE}
```

awsvpc ネットワークモードでの Fargate クラスターの AWS CloudFormation スタックの作成

```
export AWS_PROFILE=your_aws_config_profile_eg_default
export AWS_DEFAULT_REGION=your_aws_region_eg_ap-southeast-1
export ECS_CLUSTER_NAME=your_ec2_ecs_cluster_name
export ECS_LAUNCH_TYPE=FARGATE
export CREATE_IAM_ROLES=True
export ECS_CLUSTER_SECURITY_GROUP=your_security_group_eg_sg-xxxxxxxxxx
export ECS_CLUSTER_SUBNET=your_subnet_eg_subnet-xxxxxxxxxx
export ECS_TASK_ROLE_NAME=your_selected_ecs_task_role_name
export ECS_EXECUTION_ROLE_NAME=your_selected_ecs_execution_role_name

curl -O https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-
insights/latest/ecs-task-definition-templates/deployment-mode/replica-service/cwagent-
prometheus/cloudformation-quickstart/cwagent-ecs-prometheus-metric-for-awsvpc.yaml

aws cloudformation create-stack --stack-name CWAgent-Prometheus-ECS-
${ECS_CLUSTER_NAME}-${ECS_LAUNCH_TYPE}-awsvpc \
    --template-body file://cwagent-ecs-prometheus-metric-for-awsvpc.yaml \
    --parameters ParameterKey=ECSClusterName,ParameterValue=${ECS_CLUSTER_NAME} \
                 ParameterKey=CreateIAMRoles,ParameterValue=${CREATE_IAM_ROLES} \
                 ParameterKey=ECSLaunchType,ParameterValue=${ECS_LAUNCH_TYPE} \
                 ParameterKey=SecurityGroupID,ParameterValue=
${ECS_CLUSTER_SECURITY_GROUP} \
                 ParameterKey=SubnetID,ParameterValue=${ECS_CLUSTER_SUBNET} \
                 ParameterKey=TaskRoleName,ParameterValue=${ECS_TASK_ROLE_NAME} \
                 ParameterKey=ExecutionRoleName,ParameterValue=
${ECS_EXECUTION_ROLE_NAME} \
    --capabilities CAPABILITY_NAMED_IAM \
    --region ${AWS_DEFAULT_REGION} \
    --profile ${AWS_PROFILE}
```

AWS スタックによって作成された AWS CloudFormation リソース

次の表は、AWS を使用して Amazon ECS クラスターで Prometheus モニターリングを使用した Container Insights をセットアップするときに作成される AWS CloudFormation リソースの一覧です。

リソースタイプ	リソース名	コメント
AWS::SSM::Parameter	AmazonCloudWatch-CWAgentConfig-$*ECS_CLUSTER_NAME* -$*ECS_LAUNCH_TYPE* -$*ECS_NETWORK_MODE*	これは、デフォルト App Mesh および Java/JMX 埋め込みメトリクス形式定義を持つ CloudWatch エージェントです。
AWS::SSM::Parameter	AmazonCloudWatch-Prometheus ConfigName-$*ECS_CLUSTER_NAME* -$*ECS_LAUNCH_TYPE* -$*ECS_NETWORK_MODE*	これは Prometheus のスクレイピング設定です。
AWS::IAM::Role	$ECS_TASK_ROLE_NAME。	Amazon ECS タスクロール。これは、**True** に CREATE_IAM_ROLES を指定した場合にのみ作成されます 。
AWS::IAM::Role	${ECS_EXECUTION_ROLE_NAME}	Amazon ECS タスク実行ロール。これは、**True** に CREATE_IAM_ROLES を指定した場合にのみ作成されます 。
AWS::ECS::TaskDefinition	cwagent-prometheus-$*ECS_CLUSTER_NAME* -$*ECS_LOUNCH_TYPE* -$*ECS_NETWORK_MODE*	
AWS::ECS::Service	cwagent-prometheus-replica-service-$*ECS_LAUNCH_TYPE* -$*ECS_NETWORK_MODE*	

Prometheus モニターリングを使用した CloudWatch エージェントの AWS CloudFormation スタックの削除

Amazon ECS クラスターから CloudWatch エージェントを削除するには、次のコマンドを入力します。

```
export AWS_PROFILE=your_aws_config_profile_eg_default
export AWS_DEFAULT_REGION=your_aws_region_eg_ap-southeast-1
export CLOUDFORMATION_STACK_NAME=your_cloudformation_stack_name
```

```
aws cloudformation delete-stack \
--stack-name ${CLOUDFORMATION_STACK_NAME} \
--region ${AWS_DEFAULT_REGION} \
--profile ${AWS_PROFILE}
```

追加の Prometheus ソースのスクレイピングと、それらのメトリクスのインポート

Prometheus モニターリングを使用した CloudWatch エージェントは、Prometheus メトリクスをスクレイプするために 2 つの設定が必要です。1 つは標準の Prometheus 設定用で、Prometheus ドキュメントの「<scrape_config>」に記載されています。もう 1 つは CloudWatch エージェント設定用です。

Amazon ECS クラスターでは、設定は Amazon ECS タスク定義のシークレットによって AWS Systems Manager の Parameter Store と統合されます。

- このシークレット PROMETHEUS_CONFIG_CONTENT は、Prometheus スクレイプ設定用です。
- このシークレット CW_CONFIG_CONTENT は、 CloudWatch エージェント設定用です。

追加の Prometheus メトリクスソースをスクレイプし、それらのメトリクスを CloudWatch にインポートするには、Prometheus スクレイプ設定と CloudWatch エージェント設定の両方を変更し、更新された設定でエージェントを再デプロイします。

VPC セキュリティグループの要件

Prometheus ワークロードのセキュリティグループの受信ルールでは、Prometheus のメトリクスをプライベート IP でスクレイピングするために、CloudWatch エージェントへの Prometheus ポートを開く必要があります。

CloudWatch エージェントのセキュリティグループの出力ルールでは、CloudWatch エージェントがプライベート IP によって Prometheus ワークロードのポートに接続できるようにする必要があります。

Prometheus スクレイプ設定

この CloudWatch エージェントは、Prometheus のドキュメントの「<scrape_config>」に記載されているように、標準の Prometheus スクレイプ設定をサポートしています。このセクションを編集して、このファイルに既に含まれている設定を更新したり、Prometheus スクレイピングターゲットを追加したりできます。デフォルトでは、サンプル設定ファイルに次のグローバル設定行が含まれています。

```
global:
  scrape_interval: 1m
  scrape_timeout: 10s
```

- scrape_interval – ターゲットをスクレイプする頻度を定義します。
- scrape_timeout – スクレイプリクエストがタイムアウトするまでの待機時間を定義します。

また、ジョブレベルでこれらの設定に対して異なる値を定義し、グローバル設定をオーバーライドすることもできます。

Prometheus スクレイピングジョブ

CloudWatch エージェント YAML ファイルには、既にいくつかのデフォルトのスクレイピングジョブが設定されています。例えば、`cwagent-ecs-prometheus-metric-for-bridge-host.yaml` などのAmazon ECS の YAML ファイルでは、デフォルトのスクレイピングジョブが `ecs_service_discovery` セクションで設定されています。

```
"ecs_service_discovery": {
                  "sd_frequency": "1m",
                  "sd_result_file": "/tmp/cwagent_ecs_auto_sd.yaml",
                  "docker_label": {
                  },
                  "task_definition_list": [
                    {
                      "sd_job_name": "ecs-appmesh-colors",
                      "sd_metrics_ports": "9901",
                      "sd_task_definition_arn_pattern": ".*:task-definition\/.*-
ColorTeller-(white):[0-9]+",
                      "sd_metrics_path": "/stats/prometheus"
                    },
                    {
                      "sd_job_name": "ecs-appmesh-gateway",
                      "sd_metrics_ports": "9901",
                      "sd_task_definition_arn_pattern": ".*:task-definition/.*-
ColorGateway:[0-9]+",
                      "sd_metrics_path": "/stats/prometheus"
                    }
                  ]
                }
```

これらのデフォルトターゲットはそれぞれスクレイピングされ、メトリクスは埋め込みメトリクス形式を使用してログイベントで CloudWatch に送信されます。詳細については、「ログ内へのメトリクスの埋め込み」を参照してください。

Amazon ECS クラスターからのログイベントは、[/aws/ecs/containerinsights/***cluster_name***/prometheus] ロググループに保存されます。

各スクレイピングジョブは、このロググループ内の異なるログストリームに含まれています。

新しいスクレイピングターゲットを追加するには、YAML ファイルの `task_definition_list` セクションの `ecs_service_discovery` セクションに新しいエントリを追加し、エージェントを再起動します。このプロセスの例については、「新しい Prometheus スクレイピングターゲットを追加するためのチュートリアル: Prometheus API サーバーメトリクス」を参照してください。

Prometheus の CloudWatch エージェント設定

CloudWatch エージェント設定ファイルには、Prometheus スクレイピング設定の `prometheus` の `metrics_collected` セクションがあります。これには、次の設定オプションが含まれます。

- cluster_name – ログイベントのラベルとして追加されるクラスター名を指定します。このフィールドはオプションです。これを省略すると、エージェントは Amazon ECS クラスター名を検出できます。
- log_group_name – スクレイプされた Prometheus メトリクスのロググループ名を指定します。このフィールドはオプションです。これを省略すると、CloudWatch では Amazon ECS クラスターからのログに/aws/ecs/containerinsights/***cluster_name***/prometheusが使用されます。
- prometheus_config_path – Prometheus スクレイプ設定ファイルパスを指定します。このフィールドの値が `env:` で始まる場合、Prometheus スクレイプ設定ファイルの内容は、コンテナの環境変数から取得されます。このフィールドは変更しないでください。
- ecs_service_discovery – Amazon ECS Prometheus ターゲット自動検出関数の設定を指定するセクションです。Prometheus ターゲットを検出するには、コンテナの docker ラベルに基づく検出と、Amazon ECS タスク定義の ARN 正規表現に基づく検出の 2 つのモードがサポートされています。2 つのモードを一緒に使用すると、CloudWatch エージェントは {private_ip}:{port}/{metrics_path} に基づいて検出されたターゲットの重複排除を行います。

  `ecs_service_discovery` セクションには、次のフィールドを含めることができます。

  - `sd_frequency` は、Prometheus エクスポーターを検出する頻度です。数値と単位サフィックスを指定します。例えば、`1m` の場合は 1 分に 1 回、`30s` の場合は 30 秒に 1 回です。有効な単位サフィックスは、`ns`、`us`、`ms`、`s`、`m`、`h` です。

このフィールドはオプションです。デフォルト値は 60 秒 (1 分) です。

- sd_target_cluster は、自動検出のターゲット Amazon ECS クラスター名です。このフィールドはオプションです。デフォルトは、CloudWatch エージェントがインストールされている Amazon ECS クラスターの名前です。
- sd_cluster_region は、ターゲット Amazon ECS クラスターのリージョンです。このフィールドはオプションです。デフォルトは、CloudWatch エージェントがインストールされている Amazon ECS クラスターのリージョンです。
- sd_result_file は、Prometheus ターゲット結果の YAML ファイルのパスです。Prometheus スクレイプ設定は、このファイルを参照します。
- docker_label は、docker のラベルベースのサービス検出の設定を指定するために使用できるオプションのセクションです。このセクションを省略すると、docker のラベルベースの検出は使用されません。このセクションには、次のフィールドを含めることができます。
  - sd_port_label は、Prometheus メトリクスのコンテナポートを指定するコンテナの docker ラベル名です。デフォルト値は ECS_PROMETHEUS_EXPORTER_PORT です。コンテナにこの docker ラベルがない場合、CloudWatch エージェントはそれをスキップします。
  - sd_metrics_path_label は、Prometheus メトリクスパスを指定するコンテナの docker ラベル名です。デフォルト値は ECS_PROMETHEUS_METRICS_PATH です。コンテナにこの docker ラベルがない場合は、エージェントはデフォルトパス /metrics を想定します。
  - sd_job_name_label は、Prometheus スクレイプジョブ名を指定するコンテナの docker ラベル名です。デフォルト値は job です。コンテナにこの docker ラベルがない場合、CloudWatch エージェントは Prometheus スクレイプ設定でジョブ名を使用します。
- task_definition_list は、タスク定義ベースのサービス検出の設定を指定するために使用できるオプションのセクションです。このセクションを省略すると、タスク定義ベースの検出は使用されません。このセクションには、次のフィールドを含めることができます。
  - sd_task_definition_arn_pattern は、検出する Amazon ECS タスク定義を指定するために使用するパターンです。これは正規表現です。
  - sd_metrics_ports に、Prometheus メトリクスの containerPort を示します。containerPort はセミコロンで区切ります。
  - sd_container_name_pattern は、Amazon ECS タスクコンテナ名を指定します。これは正規表現です。
  - sd_metrics_path は、Prometheus のメトリクスパスを指定します。これを省略すると、エージェントはデフォルトのパス /metrics を引き受けます。

    - sd_job_name Prometheus スクレイプジョブ名を指定します。このフィールドを省略すると、CloudWatch エージェントは Prometheus スクレイプ設定のジョブ名を使用します。
  - service_name_list_for_tasks は、サービス名ベースの検出の設定を指定するために使用できるオプションのセクションです。このセクションを省略すると、サービス名ベースの検出は使用されません。このセクションには、次のフィールドを含めることができます。
    - sd_service_name_pattern は、タスクを検出する Amazon ECS サービスを指定するために使用するパターンです。これは正規表現です。
    - sd_metrics_ports は、Prometheus メトリクスの containerPort をリストします。複数の containerPorts をセミコロンで区切ります。
    - sd_container_name_pattern は、Amazon ECS タスクコンテナ名を指定します。これは正規表現です。
    - sd_metrics_path は、Prometheus のメトリクスパスを指定します。これを省略すると、エージェントはデフォルトのパス /metrics を引き受けます。
    - sd_job_name Prometheus スクレイプジョブ名を指定します。このフィールドを省略すると、CloudWatch エージェントは Prometheus スクレイプ設定のジョブ名を使用します。
- metric_declaration – 生成されるメトリクス形式が埋め込まれたログの配列を指定するセクションです。CloudWatch エージェントがインポートする各 Prometheus ソースには、デフォルトで metric_declaration セクションがあります。これらの各セクションには、次のフィールドが含まれています。
  - label_matcher は、 source_labels に表示されているラベルの値をチェックする正規表現です。一致するメトリクスは、CloudWatch に送信される埋め込みメトリクス形式に含めることができます。

    source_labels で複数のラベルを指定する場合は、^ の正規表現に $ や label_matcher 文字を使用しないことをお勧めします。
  - source_labels は、label_matcher 行によってチェックされるラベルの値を指定します。
  - label_separator は、複数の label_matcher が指定されている場合に、source_labels 行で使用するセパレータを指定します。デフォルト: ;。このデフォルトは、次の例の label_matcher 行で使用されています。
  - metric_selectors は、収集され、CloudWatch に送信されるメトリクスを指定する正規表現です。
  - dimensions は、選択した各メトリクスの CloudWatch ディメンションとして使用されるラベルのリストです。

次の metric_declaration の例を参照してください。

```
"metric_declaration": [
  {
     "source_labels":[ "Service", "Namespace"],
     "label_matcher":"(.*node-exporter.*|.*kube-dns.*);kube-system$",
     "dimensions":[
        ["Service", "Namespace"]
     ],
     "metric_selectors":[
        "^coredns_dns_request_type_count_total$"
     ]
  }
]
```

この例では、次の条件が満たされた場合にログイベントとして送信される埋め込みメトリクス形式セクションを設定します。

- Service の値には node-exporter または kube-dns が含まれます。
- Namespace の値は kube-system です。
- Prometheus メトリクス coredns_dns_request_type_count_total には、Service ラベルおよび Namespace ラベルの両方が含まれます。

送信されるログイベントには、次の強調表示されたセクションが含まれます。

```
{
   "CloudWatchMetrics":[
      {
         "Metrics":[
            {
               "Name":"coredns_dns_request_type_count_total"
            }
         ],
         "Dimensions":[
            [
               "Namespace",
               "Service"
            ]
         ],
         "Namespace":"ContainerInsights/Prometheus"
      }
```

```
    ],
    "Namespace":"kube-system",
    "Service":"kube-dns",
    "coredns_dns_request_type_count_total":2562,
    "eks_amazonaws_com_component":"kube-dns",
    "instance":"192.168.61.254:9153",
    "job":"kubernetes-service-endpoints",
    ...
}
```

Amazon ECS クラスターでの自動検出の詳細ガイド

Prometheus には、<scrape_config>で説明されているように、多数の動的なサービス検出メカニズムが用意されています。ただし、Amazon ECS の組み込みサービス検出はありません。CloudWatch エージェントはこのメカニズムを追加します。

Amazon ECS Prometheus サービス検出を有効にするときに、CloudWatch エージェントは定期的に Amazon ECS および Amazon EC2 フロントエンドに対して次の API コールを行い、ターゲット ECS クラスターで実行中の ECS タスクのメタデータを取得します。

```
EC2:DescribeInstances
ECS:ListTasks
ECS:ListServices
ECS:DescribeContainerInstances
ECS:DescribeServices
ECS:DescribeTasks
ECS:DescribeTaskDefinition
```

このメタデータは、ECS クラスター内の Prometheus ターゲットをスキャンするために CloudWatch エージェントによって使用されます。CloudWatch エージェントは、次の 3 つのサービス検出モードをサポートします。

- コンテナドッカーのラベルベースのサービス検出
- ECS タスク定義 ARN 正規表現ベースのサービス検出
- ECS サービス名正規表現ベースのサービス検出

すべてのモードを一緒に使用することができます。CloudWatch エージェントは、`{private_ip}:{port}/{metrics_path}` に基づいて検出されたターゲットの重複を解除します。

検出されたターゲットはすべて、CloudWatch エージェントコンテナ内の sd_result_file 設定フィールドで指定された結果ファイルに書き込まれます。次に、結果ファイルの例を示します。

```
- targets:
  - 10.6.1.95:32785
  labels:
    __metrics_path__: /metrics
    ECS_PROMETHEUS_EXPORTER_PORT: "9406"
    ECS_PROMETHEUS_JOB_NAME: demo-jar-ec2-bridge-dynamic
    ECS_PROMETHEUS_METRICS_PATH: /metrics
    InstanceType: t3.medium
    LaunchType: EC2
    SubnetId: subnet-123456789012
    TaskDefinitionFamily: demo-jar-ec2-bridge-dynamic-port
    TaskGroup: family:demo-jar-ec2-bridge-dynamic-port
    TaskRevision: "7"
    VpcId: vpc-01234567890
    container_name: demo-jar-ec2-bridge-dynamic-port
    job: demo-jar-ec2-bridge-dynamic
- targets:
  - 10.6.3.193:9404
  labels:
    __metrics_path__: /metrics
    ECS_PROMETHEUS_EXPORTER_PORT_SUBSET_B: "9404"
    ECS_PROMETHEUS_JOB_NAME: demo-tomcat-ec2-bridge-mapped-port
    ECS_PROMETHEUS_METRICS_PATH: /metrics
    InstanceType: t3.medium
    LaunchType: EC2
    SubnetId: subnet-123456789012
    TaskDefinitionFamily: demo-tomcat-ec2-bridge-mapped-port
    TaskGroup: family:demo-jar-tomcat-bridge-mapped-port
    TaskRevision: "12"
    VpcId: vpc-01234567890
    container_name: demo-tomcat-ec2-bridge-mapped-port
    job: demo-tomcat-ec2-bridge-mapped-port
```

この結果ファイルは、Prometheus ファイルベースのサービス検出と直接統合できます。Prometheus ファイルベースのサービス検出の詳細については、「<file_sd_config>」を参照してください。

結果ファイルが /tmp/cwagent_ecs_auto_sd.yaml に書き込まれたとします。次の Prometheus スクレイプ設定はそれを消費します。

```
global:
  scrape_interval: 1m
  scrape_timeout: 10s
scrape_configs:
  - job_name: cwagent-ecs-file-sd-config
    sample_limit: 10000
    file_sd_configs:
      - files: [ "/tmp/cwagent_ecs_auto_sd.yaml" ]
```

CloudWatch エージェントは、検出されたターゲットに対して次の追加ラベルも追加します。

- container_name
- TaskDefinitionFamily
- TaskRevision
- TaskGroup
- StartedBy
- LaunchType
- job
- __metrics_path__
- Docker のラベル

クラスターの起動タイプが EC2 の場合、次の 3 つのラベルが追加されます。

- InstanceType
- VpcId
- SubnetId

> **Note**
>
> 正規表現 [a-zA-Z_][a-zA-Z0-9_]* と一致しない Docker ラベルは除外されます。これは、Prometheus のマニュアルの「構成ファイル」 の label_name に記載されている Prometheus の規則と一致します。

ECS サービス検出の設定例

このセクションでは、ECS サービスの検出を示す例について説明します。

例 1

```
"ecs_service_discovery": {
  "sd_frequency": "1m",
  "sd_result_file": "/tmp/cwagent_ecs_auto_sd.yaml",
  "docker_label": {
  }
}
```

この例では、Docker ラベルベースのサービス検出を有効にします。CloudWatch エージェントは、ECS タスクのメタデータを 1 分間に照会し、検出されたターゲットを CloudWatch エージェントコンテナ内の `/tmp/cwagent_ecs_auto_sd.yaml` ファイルに書き込みます。

`sd_port_label` セクションの `docker_label` のデフォルト値は `ECS_PROMETHEUS_EXPORTER_PORT` です。ECS タスク内の実行中のコンテナに `ECS_PROMETHEUS_EXPORTER_PORT` ドッカーラベルがある場合、CloudWatch エージェントはその値を `container port` として使用して、コンテナのすべての公開ポートをスキャンします。一致するものがある場合、マッピングされたホストポートとコンテナのプライベート IP を使用して、Prometheus エクスポーターゲットが `private_ip:host_port` の形式で構築されます。

`sd_metrics_path_label` セクションの `docker_label` のデフォルト値は `ECS_PROMETHEUS_METRICS_PATH` です。コンテナにこのドッカーラベルがある場合、その値は `__metrics_path__` として使用されます。コンテナにこのラベルがない場合は、デフォルト値 `/metrics` が使用されます。

`sd_job_name_label` セクションの `docker_label` のデフォルト値は `job` です。コンテナにこのドッカーラベルがある場合、その値はターゲットのラベルの 1 つとして追加され、Prometheus 設定で指定されたデフォルトのジョブ名が置き換えられます。このドッカーラベルの値は、CloudWatch Logs ロググループのログストリーム名として使用されます。

例 2

```
"ecs_service_discovery": {
  "sd_frequency": "15s",
  "sd_result_file": "/tmp/cwagent_ecs_auto_sd.yaml",
  "docker_label": {
    "sd_port_label": "ECS_PROMETHEUS_EXPORTER_PORT_SUBSET_A",
```

```
    "sd_job_name_label": "ECS_PROMETHEUS_JOB_NAME"
  }
}
```

この例では、Docker ラベルベースのサービス検出を有効にします。CloudWatch エージェントは、ECS タスクのメタデータを 15 秒ごとに照会し、検出されたターゲットを CloudWatch エージェントコンテナ内の /tmp/cwagent_ecs_auto_sd.yaml ファイルに書き込みます。ドッカーラベルが ECS_PROMETHEUS_EXPORTER_PORT_SUBSET_A のコンテナがスキャンされます。ドッカーラベルが ECS_PROMETHEUS_JOB_NAME の値がジョブ名として使用されます。

例 3

```
"ecs_service_discovery": {
  "sd_frequency": "5m",
  "sd_result_file": "/tmp/cwagent_ecs_auto_sd.yaml",
  "task_definition_list": [
    {
      "sd_job_name": "java-prometheus",
      "sd_metrics_path": "/metrics",
      "sd_metrics_ports": "9404; 9406",
      "sd_task_definition_arn_pattern": ".*:task-definition/.*javajmx.*:[0-9]+"
    },
    {
      "sd_job_name": "envoy-prometheus",
      "sd_metrics_path": "/stats/prometheus",
      "sd_container_name_pattern": "^envoy$",
      "sd_metrics_ports": "9901",
      "sd_task_definition_arn_pattern": ".*:task-definition/.*appmesh.*:23"
    }
  ]
}
```

この例では、ECS タスク定義 ARN 正規表現ベースのサービス検出をイネーブルにします。CloudWatch エージェントは、ECS タスクのメタデータを 5 分ごとに照会し、検出されたターゲットを CloudWatch エージェントコンテナ内の /tmp/cwagent_ecs_auto_sd.yaml ファイルに書き込みます。

2 つのタスク定義 ARN 通常記述セクションが定義されています。

- 最初のセクションでは、ECS タスク定義 ARN に javajmx が含まれる ECS タスクが、コンテナポートスキャン用にフィルタリングされます。これらの ECS タスク内のコンテナが 9404 また

は 9406 のコンテナポートを公開する場合、マップされたホストポートとコンテナのプライベート IP が Prometheus エクスポーターターゲットの作成に使用されます。sd_metrics_path の値は __metrics_path__ から /metrics に設定します。したがって、CloudWatch エージェントは Prometheus メトリクスを private_ip:host_port/metrics からスクレイプし、スクレイプされたメトリクスは、java-prometheus ロググループ内の CloudWatch Logs の /aws/ecs/containerinsights/cluster_name/prometheus ログストリームに送信 されます。

- 2 番目のセクションでは、ECS タスク定義 ARN に appmesh があり、version の :23 がある ECS タスクが、コンテナポートスキャン用にフィルタリングされます。コンテナポートを envoy 上に公開する 9901 という名前のコンテナの場合、マッピングされたホストポートとコンテナのプライベート IP が Prometheus エクスポーターターゲットの作成に使用されます。これらの ECS タスク内の値は、9404 または 9406 のコンテナポートを公開し、マッピングされたホストポートとコンテナのプライベート IP が Prometheus エクスポーターターゲットの作成に使用されます。sd_metrics_path の値は __metrics_path__ から /stats/prometheus に設定します。したがって、CloudWatch エージェントは private_ip:host_port/stats/prometheus から Prometheus メトリクスをスクレイプし、スクレイプされたメトリクスをロググループ envoy-prometheus 内の CloudWatch Logs の /aws/ecs/containerinsights/cluster_name/prometheus ログストリームに送信します。

例 4

```
"ecs_service_discovery": {
  "sd_frequency": "5m",
  "sd_result_file": "/tmp/cwagent_ecs_auto_sd.yaml",
  "service_name_list_for_tasks": [
    {
      "sd_job_name": "nginx-prometheus",
      "sd_metrics_path": "/metrics",
      "sd_metrics_ports": "9113",
      "sd_service_name_pattern": "^nginx-.*"
    },
    {
      "sd_job_name": "haproxy-prometheus",
      "sd_metrics_path": "/stats/metrics",
      "sd_container_name_pattern": "^haproxy$",
      "sd_metrics_ports": "8404",
      "sd_service_name_pattern": ".*haproxy-service.*"
    }
  ]
}
```

この例では、ECS サービス名正規表現ベースのサービス検出を有効にします。CloudWatch エージェントは、ECS サービスのメタデータを 5 分ごとに照会し、検出されたターゲットを CloudWatch エージェントコンテナ内の /tmp/cwagent_ecs_auto_sd.yaml ファイルに書き込みます。

2 つのサービス名正規表現セクションが定義されています。

- 最初のセクションについて、正規表現 ^nginx-.* に一致する名前を持つ ECS サービスに関連付けられた ECS タスクがコンテナポートスキャン用にフィルタリングされます。これらの ECS タスク内のコンテナが 9113 のコンテナポートを公開する場合、マップされたホストポートとコンテナのプライベート IP が Prometheus エクスポーターターゲットの作成に使用されます。sd_metrics_path の値は __metrics_path__ から /metrics に設定します。したがって、CloudWatch エージェントは Prometheus メトリクスを private_ip:host_port/metrics からスクレイプし、スクレイプされたメトリクスは、nginx-prometheus ロググループ内の CloudWatch Logs の /aws/ecs/containerinsights/cluster_name/prometheus ログストリームに送信されます。
- または、2 つ目のセクションについて、正規表現 .*haproxy-service.* に一致する名前を持つ ECS サービスに関連付けられた ECS タスクがコンテナポートスキャン用にフィルタリングされます。コンテナポートを 8404 上に公開する haproxy という名前のコンテナの場合、マッピングされたホストポートとコンテナのプライベート IP が Prometheus エクスポーターターゲットの作成に使用されます。sd_metrics_path の値は __metrics_path__ から /stats/metrics に設定します。したがって、CloudWatch エージェントは Prometheus メトリクスを private_ip:host_port/stats/metrics からスクレイプし、スクレイプされたメトリクスは、haproxy-prometheus ロググループ内の CloudWatch Logs の /aws/ecs/containerinsights/cluster_name/prometheus ログストリームに送信されます。

例 5

```
"ecs_service_discovery": {
  "sd_frequency": "1m30s",
  "sd_result_file": "/tmp/cwagent_ecs_auto_sd.yaml",
  "docker_label": {
    "sd_port_label": "MY_PROMETHEUS_EXPORTER_PORT_LABEL",
    "sd_metrics_path_label": "MY_PROMETHEUS_METRICS_PATH_LABEL",
    "sd_job_name_label": "MY_PROMETHEUS_METRICS_NAME_LABEL"
  }
  "task_definition_list": [
    {
      "sd_metrics_ports": "9150",
```

```
      "sd_task_definition_arn_pattern": "*memcached.*"
    }
  ]
}
```

この例では、両方の ECS サービス検出モードを有効にします。CloudWatch エージェントは、ECS タスクのメタデータを 90 秒ごとに照会し、検出されたターゲットを CloudWatch エージェントコンテナ内の `/tmp/cwagent_ecs_auto_sd.yaml` ファイルに書き込みます。

ドッカーベースのサービス検出設定の場合。

- ドッカーラベル `MY_PROMETHEUS_EXPORTER_PORT_LABEL` 付きの ECS タスクは、Prometheus ポートスキャン用にフィルタリングされます。ターゲットの Prometheus コンテナポートは、ラベル `MY_PROMETHEUS_EXPORTER_PORT_LABEL` の値によって指定されます。
- ドッカーラベル `MY_PROMETHEUS_EXPORTER_PORT_LABEL` の値は、`__metrics_path__` に使用されます。コンテナにこのドッカーラベルがない場合は、デフォルト値 `/metrics` が使用されます。
- ドッカーラベル `MY_PROMETHEUS_EXPORTER_PORT_LABEL` の値は、ジョブラベルとして使用されます。コンテナにこのドッカーラベルがない場合は、Prometheus 設定で定義されたジョブ名が使用されます。

ECS タスク定義 ARN 正規表現ベースのサービス検出設定の場合。

- ECS タスク定義 ARN `memcached` 内の ECS タスクは、コンテナポートスキャン用にフィルタリングされます。ターゲットの Prometheus コンテナポートは、`sd_metrics_ports` で定義されている 9150 です。デフォルトのメトリクスパス `/metrics` が使用されます。Prometheus 設定で定義されているジョブ名が使用されます。

(オプション) Prometheus メトリクスのテストのためにコンテナ化された Amazon ECS のサンプルワークロードを設定する

CloudWatch Container Insights で Prometheus メトリクスのサポートをテストするには、次のコンテナ化されたワークロードを 1 つ以上設定できます。Prometheus をサポートする CloudWatch エージェントは、これらの各ワークロードからメトリクスを自動的に収集します。デフォルトで収集されるメトリクスを表示する方法については、「CloudWatch エージェントにより収集される Prometheus メトリクス」を参照してください。

トピック

- Amazon ECS クラスターの App Mesh ワークロードのサンプル
- Amazon ECS クラスター用の Java/JMX ワークロードのサンプル
- Amazon ECS クラスターの NGINX ワークロードのサンプル
- Amazon ECS クラスターの NGINX Plus ワークロードのサンプル
- 新しい Prometheus スクレイピングターゲットを追加するためのチュートリアル: Amazon ECS の Memcached
- Amazon ECS Fargate の Redis Prometheus メトリクスをスクレイプするためのチュートリアル

Amazon ECS クラスターの App Mesh ワークロードのサンプル

Amazon ECS のサンプル Prometheus ワークロードからメトリクスを収集するには、クラスターで Container Insights を実行している必要があります。Container Insights のインストールの詳細については、「Amazon ECS での Container Insights のセットアップ」を参照してください。

まず、このチュートリアルに従って、サンプルカラーアプリを Amazon ECS クラスターにデプロイします。完了したら、ポート 9901 に App Mesh Prometheus メトリクスが公開されます。

次に、以下のステップに従って、カラーアプリをインストールした同じ Amazon ECS クラスターに Prometheus モニターリングを使用して CloudWatch エージェントをインストールします。このセクションのステップでは、ブリッジネットワークモードで CloudWatch エージェントをインストールします。

チュートリアルで設定した環境変数 `ENVIRONMENT_NAME`、`AWS_PROFILE`、`AWS_DEFAULT_REGION` は、次のステップでも使用されます。

テスト用に Prometheus モニターリングを使用して CloudWatch エージェントをインストールするには

1. 次のコマンドを入力して、AWS CloudFormation テンプレートをダウンロードします。

```
curl -O https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-insights/latest/ecs-task-definition-templates/deployment-mode/replica-service/cwagent-prometheus/cloudformation-quickstart/cwagent-ecs-prometheus-metric-for-bridge-host.yaml
```

2. 次のコマンドを入力して、ネットワークモードを設定します。

```
export ECS_CLUSTER_NAME=${ENVIRONMENT_NAME}
export ECS_NETWORK_MODE=bridge
```

3. 次のコマンドを入力して、AWS CloudFormation スタックを作成します。

```
aws cloudformation create-stack --stack-name CWAgent-Prometheus-ECS-
${ECS_CLUSTER_NAME}-EC2-${ECS_NETWORK_MODE} \
    --template-body file://cwagent-ecs-prometheus-metric-for-bridge-host.yaml \
    --parameters ParameterKey=ECSClusterName,ParameterValue=${ECS_CLUSTER_NAME} \
                 ParameterKey=CreateIAMRoles,ParameterValue=True \
                 ParameterKey=ECSNetworkMode,ParameterValue=${ECS_NETWORK_MODE} \
                 ParameterKey=TaskRoleName,ParameterValue=CWAgent-Prometheus-
TaskRole-${ECS_CLUSTER_NAME} \
                 ParameterKey=ExecutionRoleName,ParameterValue=CWAgent-Prometheus-
ExecutionRole-${ECS_CLUSTER_NAME} \
    --capabilities CAPABILITY_NAMED_IAM \
    --region ${AWS_DEFAULT_REGION} \
    --profile ${AWS_PROFILE}
```

4. (オプション) AWS CloudFormation スタックが作成されると、 CREATE_COMPLETE メッセージが表示されます。そのメッセージが表示される前にステータスを確認する場合は、次のコマンドを入力します。

```
aws cloudformation describe-stacks \
--stack-name CWAgent-Prometheus-ECS-${ECS_CLUSTER_NAME}-EC2-${ECS_NETWORK_MODE} \
--query 'Stacks[0].StackStatus' \
--region ${AWS_DEFAULT_REGION} \
--profile ${AWS_PROFILE}
```

トラブルシューティング

チュートリアルのステップでは、jq を使用して、 の出力結果を解析しますAWS CLI jq のインストールの詳細については、「jq」を参照してください。jq が正しく解析できるように、次のコマンドを使用して AWS CLI のデフォルトの出力形式を JSON に設定します。

```
$ aws configure
```

レスポンスが `Default output format` に到達したら、**`json`** と入力します 。

Prometheus モニターリングを使用した CloudWatch エージェントのアンインストール

テストが終了したら、次のコマンドを入力し、AWS CloudFormation スタックを削除して CloudWatch エージェントをアンインストールします。

```
aws cloudformation delete-stack \
--stack-name CWAgent-Prometheus-ECS-${ECS_CLUSTER_NAME}-EC2-${ECS_NETWORK_MODE} \
--region ${AWS_DEFAULT_REGION} \
--profile ${AWS_PROFILE}
```

Amazon ECS クラスター用の Java/JMX ワークロードのサンプル

JMX Exporter は、Prometheus メトリクスとして JMX mBeans をスクレイプおよび公開できる公式の Prometheus エクスポーターです。詳細については、prometheus/jmx_exporter を参照してください。

Prometheus をサポートする CloudWatch エージェントは、Amazon ECS クラスターのサービス検出設定に基づいて Java/JMX Prometheus メトリクスをスクレイプします。別のポートまたは metrics_path でメトリクスを公開するように JMX Exporter を設定できます。ポートまたはパスを変更する場合は、CloudWatch エージェント設定のデフォルトの `ecs_service_discovery` セクションを更新します。

Amazon ECS のサンプル Prometheus ワークロードからメトリクスを収集するには、クラスターで Container Insights を実行している必要があります。Container Insights のインストールの詳細については、「Amazon ECS での Container Insights のセットアップ」を参照してください。

Amazon ECS クラスターの Java/JMX サンプルワークロードをインストールするには

1. これらのセクションのステップに従って、Docker イメージを作成します。
   - 例: Prometheus メトリクスを使用した Java Jar アプリケーション Docker イメージ
   - 例: Prometheus メトリクスを使用した Apache Tomcat Docker イメージ
2. Amazon ECS タスク定義ファイルに次の 2 つの docker ラベルを指定します。その後、クラスター内で Amazon ECS サービスまたは Amazon ECS タスクとしてタスク定義を実行できます。
   - Prometheus メトリクスが公開されている ContainerPort を指定するようにECS_PROMETHEUS_EXPORTER_PORT を設定します 。
   - `Java_EMF_Metrics` を `true` に設定します。CloudWatch エージェントは、このフラグを使用してログイベントに埋め込みメトリクス形式を生成します。

   次に例を示します。

```
{
  "family": "workload-java-ec2-bridge",
  "taskRoleArn": "{{task-role-arn}}",
  "executionRoleArn": "{{execution-role-arn}}",
  "networkMode": "bridge",
  "containerDefinitions": [
    {
      "name": "tomcat-prometheus-workload-java-ec2-bridge-dynamic-port",
      "image": "your_docker_image_tag_for_tomcat_with_prometheus_metrics",
      "portMappings": [
        {
          "hostPort": 0,
          "protocol": "tcp",
          "containerPort": 9404
        }
      ],
      "dockerLabels": {
        "ECS_PROMETHEUS_EXPORTER_PORT": "9404",
        "Java_EMF_Metrics": "true"
      }
    }
  ],
  "requiresCompatibilities": [
    "EC2"  ],
  "cpu": "256",
  "memory": "512"
  }
```

AWS CloudFormation テンプレートの CloudWatch エージェントのデフォルト設定では、docker のラベルベースのサービス検出とタスク定義の ARN ベースのサービス検出の両方が有効になります。これらのデフォルト設定を表示するには、「CloudWatch エージェント YAML 設定ファイル」の 65 行目を参照してください。ECS_PROMETHEUS_EXPORTER_PORT ラベル付きのコンテナは、Prometheus スクレイピング用に指定されたコンテナポートに基づいて自動検出されます。

CloudWatch エージェントのデフォルト設定には、同じファイルの 112 行目に Java/JMX の metric_declaration 設定もあります。ターゲットコンテナのすべての docker ラベルは、Prometheus メトリクスに追加ラベルとして追加され、CloudWatch Logs に送信されます。docker ラベル Java_EMF_Metrics="true" 付きの Java/JMX コンテナの場合、埋め込みメトリクス形式が生成されます。

Amazon ECS クラスターの NGINX ワークロードのサンプル

NGINX Prometheus エクスポーターは、NGINX データを Prometheus のメトリクスとしてスクレイピングして公開できます。この例では、エクスポーターを Amazon ECS 用の NGINX リバースプロキシサービスと連動させて使用します。

NGINX Prometheus エクスポーターの詳細については、GitHub の nginx-prometheus-exporter をご参照ください。NGINX リバースプロキシの詳細については、GitHub の ecs-nginx-reverse-proxyをご参照ください。

Prometheus をサポートする CloudWatch エージェントは、Amazon ECS クラスターのサービス検出設定に基づいて NGINX Prometheus メトリクスをスクレイプします。別のポートまたはパスでメトリクスを公開するように NGINX Prometheus Exporter を設定できます。ポートまたはパスを変更する場合は、CloudWatch エージェント設定ファイルの `ecs_service_discovery` セクションを更新します。

Amazon ECS クラスター用の NGINX リバースプロキシサンプルワークロードをインストールする

NGINX リバースプロキシサンプルワークロードをインストールするには、次の手順を実行します。

Docker イメージを作成する

NGINX リバースプロキシサンプルワークロード用の Docker イメージを作成するには

1. NGINX リバースプロキシリポジトリから次のフォルダをダウンロードします: https://github.com/awslabs/ecs-nginx-reverse-proxy/tree/master/reverse-proxy/。
2. app ディレクトリを検索し、そのディレクトリからイメージを作成します。

   ```
   docker build -t web-server-app ./path-to-app-directory
   ```

3. NGINX 用のカスタムイメージを作成します。まず、次の 2 つのファイルを含むディレクトリを作成します。

   - サンプル Dockerfile:

     ```
     FROM nginx
     COPY nginx.conf /etc/nginx/nginx.conf
     ```

   - https://github.com/awslabs/ecs-nginx-reverse-proxy/tree/master/reverse-proxy/ から変更された `nginx.conf` ファイル:

```
events {
  worker_connections 768;
}

http {
  # Nginx will handle gzip compression of responses from the app server
  gzip on;
  gzip_proxied any;
  gzip_types text/plain application/json;
  gzip_min_length 1000;

  server{
    listen 8080;
    location /stub_status {
        stub_status   on;
    }
  }

  server {
    listen 80;

    # Nginx will reject anything not matching /api
    location /api {
      # Reject requests with unsupported HTTP method
      if ($request_method !~ ^(GET|POST|HEAD|OPTIONS|PUT|DELETE)$) {
        return 405;
      }

      # Only requests matching the whitelist expectations will
      # get sent to the application server
      proxy_pass http://app:3000;
      proxy_http_version 1.1;
      proxy_set_header Upgrade $http_upgrade;
      proxy_set_header Connection 'upgrade';
      proxy_set_header Host $host;
      proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
      proxy_cache_bypass $http_upgrade;
    }
  }
}
```

> **Note**
>
> stub_status は、nginx-prometheus-exporter がメトリクスをスクレイピングするように設定されているのと同じポートで有効にする必要があります。このタスク定義例では、nginx-prometheus-exporter がポート 8080 からメトリクスをスクレイピングするように設定されています。

4. 新しいディレクトリのファイルからイメージを作成します。

```
docker build -t nginx-reverse-proxy ./path-to-your-directory
```

5. 後で使用するために、新しいイメージをイメージリポジトリにアップロードします。

Amazon ECS で NGINX とウェブサーバーアプリを実行するタスク定義を作成します。

次に、タスク定義を設定します。

このタスク定義により、NGINX Prometheus メトリクスの収集とエクスポートが可能になります。NGINX コンテナは、アプリからの入力を追跡し、(nginx.conf で設定されているとおり) そのデータをポート 8080 に公開します。NGINX Prometheus エクスポーターコンテナは、CloudWatch で使用するために、これらのメトリクスをスクレイピングし、ポート 9113 に投稿します。

NGINX サンプル Amazon ECS ワークロードのタスク定義を設定するには

1. 次の内容でタスク定義 JSON ファイルを作成します。*your-customized-nginx-iamge* をカスタマイズした NGINX イメージのイメージ URI に置き換え、*your-web-server-app-image* をウェブサーバーアプリイメージのイメージ URI に置き換えます。

```
{
  "containerDefinitions": [
    {
      "name": "nginx",
      "image": "your-customized-nginx-image",
      "memory": 256,
      "cpu": 256,
      "essential": true,
      "portMappings": [
        {
          "containerPort": 80,
          "protocol": "tcp"
```

```
        }
      ],
      "links": [
        "app"
      ]
    },
    {
      "name": "app",
      "image": "your-web-server-app-image",
      "memory": 256,
      "cpu": 256,
      "essential": true
    },
    {
      "name": "nginx-prometheus-exporter",
      "image": "docker.io/nginx/nginx-prometheus-exporter:0.8.0",
      "memory": 256,
      "cpu": 256,
      "essential": true,
      "command": [
        "-nginx.scrape-uri",
        "http://nginx:8080/stub_status"
    ],
    "links":[
      "nginx"
    ],
      "portMappings":[
        {
          "containerPort": 9113,
          "protocol": "tcp"
        }
      ]
    }
  ],
  "networkMode": "bridge",
  "placementConstraints": [],
  "family": "nginx-sample-stack"
}
```

2. 次のコマンドを入力して、タスク定義を登録します。

```
aws ecs register-task-definition --cli-input-json file://path-to-your-task-definition-json
```

3. 次のコマンドを入力して、タスクを実行するサービスを作成します。

   サービス名は変更しないでください。CloudWatch エージェントサービスは、タスクを開始したサービスの名前パターンを使用してタスクを検索する設定を使用して実行します。例えば、このコマンドによって起動されたタスクを CloudWatch エージェントが検出するには、sd_service_name_pattern の値を ^nginx-service$ に指定します。次のセクションでは、詳細を説明します。

```
aws ecs create-service \
 --cluster your-cluster-name \
 --service-name nginx-service \
 --task-definition nginx-sample-stack:1 \
 --desired-count 1
```

NGINX Prometheus メトリクスをスクレイプするように CloudWatch エージェントを設定する

最後のステップでは、NGINX メトリクスをスクレイピングするように CloudWatch エージェントを設定します。この例では、CloudWatch エージェントはサービス名パターンとポート 9113 を使用してタスクを検出します。このポートでは、エクスポーターは NGINX の Prometheus メトリクスを公開します。タスクが検出され、メトリクスが使用可能になると、CloudWatch エージェントは、ログストリーム nginx-prometheus-exporter への収集したメトリクスの投稿を開始します。

NGINX メトリクスをスクレイピングするように CloudWatch エージェントを設定するには

1. 次のコマンドを入力して、必要な YAML ファイルの最新バージョンをダウンロードします。

```
curl -O https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-insights/latest/ecs-task-definition-templates/deployment-mode/replica-service/cwagent-prometheus/cloudformation-quickstart/cwagent-ecs-prometheus-metric-for-bridge-host.yaml
```

2. テキストエディタでファイルを開き、value セクションの resource:CWAgentConfigSSMParameter キーにある完全な CloudWatch エージェント設定を見つけます。その後、ecs_service_discovery セクションで、次の service_name_list_for_tasks セクションを追加します。

```
"service_name_list_for_tasks": [
  {
    "sd_job_name": "nginx-prometheus-exporter",
    "sd_metrics_path": "/metrics",
```

```
    "sd_metrics_ports": "9113",
    "sd_service_name_pattern": "^nginx-service$"
   }
],
```

3. 同じファイルで、metric_declaration セクションに次のセクションを追加して、NGINX メトリクスを許可します。デフォルトのインデントパターンに従ってください。

```
{
  "source_labels": ["job"],
  "label_matcher": ".*nginx.*",
  "dimensions": [["ClusterName", "TaskDefinitionFamily", "ServiceName"]],
  "metric_selectors": [
    "^nginx_.*$"
  ]
},
```

4. このクラスターにまだ CloudWatch エージェントがデプロイされていない場合は、ステップ 8 に進みます。

   AWS CloudFormation を使用することによって Amazon ECS クラスターに CloudWatch エージェントを既にデプロイしている場合は、次のコマンドを入力して変更セットを作成できます。

```
ECS_CLUSTER_NAME=your_cluster_name
AWS_REGION=your_aws_region
ECS_NETWORK_MODE=bridge
CREATE_IAM_ROLES=True
ECS_TASK_ROLE_NAME=your_selected_ecs_task_role_name
ECS_EXECUTION_ROLE_NAME=your_selected_ecs_execution_role_name

aws cloudformation create-change-set --stack-name CWAgent-Prometheus-ECS-
${ECS_CLUSTER_NAME}-EC2-${ECS_NETWORK_MODE} \
    --template-body file://cwagent-ecs-prometheus-metric-for-bridge-host.yaml \
    --parameters ParameterKey=ECSClusterName,ParameterValue=$ECS_CLUSTER_NAME \
                 ParameterKey=CreateIAMRoles,ParameterValue=$CREATE_IAM_ROLES \
                 ParameterKey=ECSNetworkMode,ParameterValue=$ECS_NETWORK_MODE \
                 ParameterKey=TaskRoleName,ParameterValue=$ECS_TASK_ROLE_NAME \
                 ParameterKey=ExecutionRoleName,ParameterValue=
$ECS_EXECUTION_ROLE_NAME \
    --capabilities CAPABILITY_NAMED_IAM \
    --region $AWS_REGION \
    --change-set-name nginx-scraping-support
```

5. https://console.aws.amazon.com/cloudformation で AWS CloudFormation コンソール を開きます。
6. 新しく作成されたチェンジセット nginx-scraping-support を確認します。CWAgentConfigSSMParameter リソースに適用された変更が 1 つ表示されます。次のコマンドを入力して、変更セットを実行し、CloudWatch エージェントタスクを再起動します。

```
aws ecs update-service --cluster $ECS_CLUSTER_NAME \
--desired-count 0 \
--service cwagent-prometheus-replica-service-EC2-$ECS_NETWORK_MODE \
--region $AWS_REGION
```

7. 10 秒ほど待ってから、次のコマンドを入力します。

```
aws ecs update-service --cluster $ECS_CLUSTER_NAME \
--desired-count 1 \
--service cwagent-prometheus-replica-service-EC2-$ECS_NETWORK_MODE \
--region $AWS_REGION
```

8. クラスターの Prometheus メトリクス収集を使用して CloudWatch エージェントを初めてインストールする場合は、次のコマンドを入力します。

```
ECS_CLUSTER_NAME=your_cluster_name
AWS_REGION=your_aws_region
ECS_NETWORK_MODE=bridge
CREATE_IAM_ROLES=True
ECS_TASK_ROLE_NAME=your_selected_ecs_task_role_name
ECS_EXECUTION_ROLE_NAME=your_selected_ecs_execution_role_name

aws cloudformation create-stack --stack-name CWAgent-Prometheus-ECS-
${ECS_CLUSTER_NAME}-EC2-${ECS_NETWORK_MODE} \
    --template-body file://cwagent-ecs-prometheus-metric-for-bridge-host.yaml \
    --parameters ParameterKey=ECSClusterName,ParameterValue=$ECS_CLUSTER_NAME \
                 ParameterKey=CreateIAMRoles,ParameterValue=$CREATE_IAM_ROLES \
                 ParameterKey=ECSNetworkMode,ParameterValue=$ECS_NETWORK_MODE \
                 ParameterKey=TaskRoleName,ParameterValue=$ECS_TASK_ROLE_NAME \
                 ParameterKey=ExecutionRoleName,ParameterValue=
$ECS_EXECUTION_ROLE_NAME \
    --capabilities CAPABILITY_NAMED_IAM \
    --region $AWS_REGION
```

NGINX メトリクスとログの表示

これで、収集されている NGINX メトリクスを表示できるようになりました。

サンプル NGINX ワークロードのメトリクスを表示するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. クラスターが実行されているリージョンにおいて、左のナビゲーションペインで [Metrics] (メトリクス) を選択します。ContainerInsights/Prometheus 名前空間を検索して、メトリクスを確認します。
3. CloudWatch Logs イベントを表示するには、ナビゲーションペインで [Log Groups (ロググループ)] を選択します。イベントは、ログストリーム *`nginx-prometheus-exporter`* のロググループ /aws/containerinsights/***`your_cluster_name`***/prometheus にあります。

Amazon ECS クラスターの NGINX Plus ワークロードのサンプル

NGINX Plus は NGINX の商用バージョンです。これを使用するには、ライセンスが必要です。詳細については、「NGINX Plus」をご参照ください。

NGINX Prometheus エクスポーターは、NGINX データを Prometheus のメトリクスとしてスクレイピングして公開できます。この例では、エクスポーターを Amazon ECS 用の NGINX Plus リバースプロキシサービスと連動させて使用します。

NGINX Prometheus エクスポーターの詳細については、GitHub の nginx-prometheus-exporter をご参照ください。NGINX リバースプロキシの詳細については、GitHub の ecs-nginx-reverse-proxyをご参照ください。

Prometheus をサポートする CloudWatch エージェントは、Amazon ECS クラスターのサービス検出設定に基づいて NGINX Plus Prometheus メトリクスをスクレイプします。別のポートまたはパスでメトリクスを公開するように NGINX Prometheus Exporter を設定できます。ポートまたはパスを変更する場合は、CloudWatch エージェント設定ファイルの `ecs_service_discovery` セクションを更新します。

Amazon ECS クラスター用の NGINX Plus リバースプロキシサンプルワークロードをインストールする

NGINX リバースプロキシサンプルワークロードをインストールするには、次の手順を実行します。

Docker イメージを作成する

NGINX Plus リバースプロキシサンプルワークロード用の Docker イメージを作成するには

1. NGINX リバースプロキシリポジトリから次のフォルダをダウンロードします: https://github.com/awslabs/ecs-nginx-reverse-proxy/tree/master/reverse-proxy/。
2. app ディレクトリを検索し、そのディレクトリからイメージを作成します。

   ```
   docker build -t web-server-app ./path-to-app-directory
   ```

3. NGINX Plus 用のカスタムイメージを作成します。NGINX Plus のイメージを作成する前に、ライセンスされている NGINX Plus の `nginx-repo.key` という名前のキーと SSL 証明書 `nginx-repo.crt` を取得する必要があります。ディレクトリを作成し、その中に `nginx-repo.key` ファイルと `nginx-repo.crt` ファイルを格納します。

   作成したディレクトリに、次の 2 つのファイルを作成します。

   - 以下の内容を含むサンプル Dockerfile。この docker ファイルは https://docs.nginx.com/nginx/admin-guide/installing-nginx/installing-nginx-docker/#docker_plus_image で提供されているサンプルファイルから採用されています。実行する重要な変更は、次のステップで作成される `nginx.conf` と呼ばれる別個のファイルをロードすることです。

     ```
     FROM debian:buster-slim

     LABEL maintainer="NGINX Docker Maintainers <docker-maint@nginx.com>“

     # Define NGINX versions for NGINX Plus and NGINX Plus modules
     # Uncomment this block and the versioned nginxPackages block in the main RUN
     # instruction to install a specific release
     # ENV NGINX_VERSION 21
     # ENV NJS_VERSION 0.3.9
     # ENV PKG_RELEASE 1~buster

     # Download certificate and key from the customer portal (https://cs.nginx.com
      (https://cs.nginx.com/))
     # and copy to the build context
     COPY nginx-repo.crt /etc/ssl/nginx/
     COPY nginx-repo.key /etc/ssl/nginx/
     # COPY nginx.conf /etc/ssl/nginx/nginx.conf

     RUN set -x \
     ```

```
# Create nginx user/group first, to be consistent throughout Docker variants
&& addgroup --system --gid 101 nginx \
&& adduser --system --disabled-login --ingroup nginx --no-create-home --home /
nonexistent --gecos "nginx user" --shell /bin/false --uid 101 nginx \
&& apt-get update \
&& apt-get install --no-install-recommends --no-install-suggests -y ca-
certificates gnupg1 \
&& \
NGINX_GPGKEY=573BFD6B3D8FBC641079A6ABABF5BD827BD9BF62; \
found=''; \
for server in \
ha.pool.sks-keyservers.net (http://ha.pool.sks-keyservers.net/) \
hkp://keyserver.ubuntu.com:80 \
hkp://p80.pool.sks-keyservers.net:80 \
pgp.mit.edu (http://pgp.mit.edu/) \
; do \
echo "Fetching GPG key $NGINX_GPGKEY from $server"; \
apt-key adv --keyserver "$server" --keyserver-options timeout=10 --recv-keys
 "$NGINX_GPGKEY" && found=yes && break; \
done; \
test -z "$found" && echo >&2 "error: failed to fetch GPG key $NGINX_GPGKEY" &&
 exit 1; \
apt-get remove --purge --auto-remove -y gnupg1 && rm -rf /var/lib/apt/lists/* \
# Install the latest release of NGINX Plus and/or NGINX Plus modules
# Uncomment individual modules if necessary
# Use versioned packages over defaults to specify a release
&& nginxPackages=" \
nginx-plus \
# nginx-plus=${NGINX_VERSION}-${PKG_RELEASE} \
# nginx-plus-module-xslt \
# nginx-plus-module-xslt=${NGINX_VERSION}-${PKG_RELEASE} \
# nginx-plus-module-geoip \
# nginx-plus-module-geoip=${NGINX_VERSION}-${PKG_RELEASE} \
# nginx-plus-module-image-filter \
# nginx-plus-module-image-filter=${NGINX_VERSION}-${PKG_RELEASE} \
# nginx-plus-module-perl \
# nginx-plus-module-perl=${NGINX_VERSION}-${PKG_RELEASE} \
# nginx-plus-module-njs \
# nginx-plus-module-njs=${NGINX_VERSION}+${NJS_VERSION}-${PKG_RELEASE} \
" \
&& echo "Acquire::https::plus-pkgs.nginx.com::Verify-Peer \"true\";" >> /etc/apt/
apt.conf.d/90nginx \
&& echo "Acquire::https::plus-pkgs.nginx.com::Verify-Host \"true\";" >> /etc/apt/
apt.conf.d/90nginx \
```

```
&& echo "Acquire::https::plus-pkgs.nginx.com::SslCert \"/etc/ssl/nginx/nginx-
repo.crt\";" >> /etc/apt/apt.conf.d/90nginx \
&& echo "Acquire::https::plus-pkgs.nginx.com::SslKey \"/etc/ssl/nginx/nginx-
repo.key\";" >> /etc/apt/apt.conf.d/90nginx \
&& printf "deb https://plus-pkgs.nginx.com/debian buster nginx-plus\n" > /etc/
apt/sources.list.d/nginx-plus.list \
&& apt-get update \
&& apt-get install --no-install-recommends --no-install-suggests -y \
$nginxPackages \
gettext-base \
curl \
&& apt-get remove --purge --auto-remove -y && rm -rf /var/lib/apt/lists/* /etc/
apt/sources.list.d/nginx-plus.list \
&& rm -rf /etc/apt/apt.conf.d/90nginx /etc/ssl/nginx

# Forward request logs to Docker log collector
RUN ln -sf /dev/stdout /var/log/nginx/access.log \
&& ln -sf /dev/stderr /var/log/nginx/error.log

COPY nginx.conf /etc/nginx/nginx.conf

EXPOSE 80

STOPSIGNAL SIGTERM

CMD ["nginx", "-g", "daemon off;"]
```

- https://github.com/awslabs/ecs-nginx-reverse-proxy/tree/master/reverse-proxy/nginx から変更された nginx.conf ファイル:

```
events {
  worker_connections 768;
}

http {
  # Nginx will handle gzip compression of responses from the app server
  gzip on;
  gzip_proxied any;
  gzip_types text/plain application/json;
  gzip_min_length 1000;

  upstream backend {
    zone name 10m;
```

```
    server app:3000    weight=2;
    server app2:3000    weight=1;
  }

  server{
    listen 8080;
    location /api {
      api write=on;
    }
  }

  match server_ok {
    status 100-599;
  }

  server {
    listen 80;
    status_zone zone;
    # Nginx will reject anything not matching /api
    location /api {
      # Reject requests with unsupported HTTP method
      if ($request_method !~ ^(GET|POST|HEAD|OPTIONS|PUT|DELETE)$) {
        return 405;
      }

      # Only requests matching the whitelist expectations will
      # get sent to the application server
      proxy_pass http://backend;
      health_check uri=/lorem-ipsum match=server_ok;
      proxy_http_version 1.1;
      proxy_set_header Upgrade $http_upgrade;
      proxy_set_header Connection 'upgrade';
      proxy_set_header Host $host;
      proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
      proxy_cache_bypass $http_upgrade;
    }
  }
}
```

4. 新しいディレクトリのファイルからイメージを作成します。

```
docker build -t nginx-plus-reverse-proxy ./path-to-your-directory
```

5. 後で使用するために、新しいイメージをイメージリポジトリにアップロードします。

Amazon ECS で NGINX Plus とウェブサーバーアプリを実行するタスク定義を作成します。

次に、タスク定義を設定します。

このタスク定義により、NGINX Plus Prometheus メトリクスの収集とエクスポートが可能になります。NGINX コンテナは、アプリからの入力を追跡し、(nginx.conf で設定されているとおり) そのデータをポート 8080 に公開します。NGINX Prometheus エクスポーターコンテナは、CloudWatch で使用するために、これらのメトリクスをスクレイピングし、ポート 9113 に投稿します。

NGINX サンプル Amazon ECS ワークロードのタスク定義を設定するには

1. 次の内容でタスク定義 JSON ファイルを作成します。*your-customized-nginx-plus-image* をカスタマイズした NGINX Plus イメージのイメージ URI に置き換え、*your-web-server-app-image* をウェブサーバーアプリイメージのイメージ URI に置き換えます。

```
{
  "containerDefinitions": [
    {
      "name": "nginx",
      "image": "your-customized-nginx-plus-image",
      "memory": 256,
      "cpu": 256,
      "essential": true,
      "portMappings": [
        {
          "containerPort": 80,
          "protocol": "tcp"
        }
      ],
      "links": [
        "app",
        "app2"
      ]
    },
    {
      "name": "app",
      "image": "your-web-server-app-image",
      "memory": 256,
      "cpu": 128,
      "essential": true
    },
    {
      "name": "app2",
```

```
      "image": "your-web-server-app-image",
      "memory": 256,
      "cpu": 128,
      "essential": true
    },
    {
      "name": "nginx-prometheus-exporter",
      "image": "docker.io/nginx/nginx-prometheus-exporter:0.8.0",
      "memory": 256,
      "cpu": 256,
      "essential": true,
      "command": [
        "-nginx.plus",
        "-nginx.scrape-uri",
         "http://nginx:8080/api"
    ],
    "links":[
      "nginx"
    ],
      "portMappings":[
        {
          "containerPort": 9113,
          "protocol": "tcp"
        }
      ]
    }
  ],
  "networkMode": "bridge",
  "placementConstraints": [],
  "family": "nginx-plus-sample-stack"
}
```

2. タスク定義を登録します。

```
aws ecs register-task-definition --cli-input-json file://path-to-your-task-
definition-json
```

3. 次のコマンドを入力して、タスクを実行するサービスを作成します。

```
aws ecs create-service \
 --cluster your-cluster-name \
 --service-name nginx-plus-service \
 --task-definition nginx-plus-sample-stack:1 \
```

```
  --desired-count 1
```

サービス名は変更しないでください。CloudWatch エージェントサービスは、タスクを開始したサービスの名前パターンを使用してタスクを検索する設定を使用して実行します。例えば、このコマンドによって起動されたタスクを CloudWatch エージェントが検出するには、sd_service_name_pattern の値を ^nginx-plus-service$ に指定します。次のセクションでは、詳細を説明します。

NGINX Plus Prometheus メトリクスをスクレイプするように CloudWatch エージェントを設定する

最後のステップでは、NGINX メトリクスをスクレイピングするように CloudWatch エージェントを設定します。この例では、CloudWatch エージェントはサービス名パターンとポート 9113 を使用してタスクを検出します。このポートでは、エクスポーターは NGINX の Prometheus メトリクスを公開します。タスクが検出され、メトリクスが使用可能になると、CloudWatch エージェントは、ログストリーム nginx-prometheus-exporter への収集したメトリクスの投稿を開始します。

NGINX メトリクスをスクレイピングするように CloudWatch エージェントを設定するには

1. 次のコマンドを入力して、必要な YAML ファイルの最新バージョンをダウンロードします。

```
curl -O https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-insights/latest/ecs-task-definition-templates/deployment-mode/replica-service/cwagent-prometheus/cloudformation-quickstart/cwagent-ecs-prometheus-metric-for-bridge-host.yaml
```

2. テキストエディタでファイルを開き、value セクションの resource:CWAgentConfigSSMParameter キーにある完全な CloudWatch エージェント設定を見つけます。その後、ecs_service_discovery セクションで、次の service_name_list_for_tasks セクションを追加します。

```
"service_name_list_for_tasks": [
  {
    "sd_job_name": "nginx-plus-prometheus-exporter",
    "sd_metrics_path": "/metrics",
    "sd_metrics_ports": "9113",
    "sd_service_name_pattern": "^nginx-plus.*"
   }
],
```

3. 同じファイルで、metric_declaration セクションに次のセクションを追加して、NGINX Plus メトリクスを許可します。デフォルトのインデントパターンに従ってください。

```
{
  "source_labels": ["job"],
  "label_matcher": "^nginx-plus.*",
  "dimensions": [["ClusterName", "TaskDefinitionFamily", "ServiceName"]],
  "metric_selectors": [
    "^nginxplus_connections_accepted$",
    "^nginxplus_connections_active$",
    "^nginxplus_connections_dropped$",
    "^nginxplus_connections_idle$",
    "^nginxplus_http_requests_total$",
    "^nginxplus_ssl_handshakes$",
    "^nginxplus_ssl_handshakes_failed$",
    "^nginxplus_up$",
    "^nginxplus_upstream_server_health_checks_fails$"
  ]
},
{
  "source_labels": ["job"],
  "label_matcher": "^nginx-plus.*",
  "dimensions": [["ClusterName", "TaskDefinitionFamily", "ServiceName",
 "upstream"]],
  "metric_selectors": [
    "^nginxplus_upstream_server_response_time$"
  ]
},
{
  "source_labels": ["job"],
  "label_matcher": "^nginx-plus.*",
  "dimensions": [["ClusterName", "TaskDefinitionFamily", "ServiceName", "code"]],
  "metric_selectors": [
    "^nginxplus_upstream_server_responses$",
    "^nginxplus_server_zone_responses$"
  ]
},
```

4. このクラスターにまだ CloudWatch エージェントがデプロイされていない場合は、ステップ 8 に進みます。

   AWS CloudFormation を使用することによって Amazon ECS クラスターに CloudWatch エージェントを既にデプロイしている場合は、次のコマンドを入力して変更セットを作成できます。

```
ECS_CLUSTER_NAME=your_cluster_name
AWS_REGION=your_aws_region
ECS_NETWORK_MODE=bridge
CREATE_IAM_ROLES=True
ECS_TASK_ROLE_NAME=your_selected_ecs_task_role_name
ECS_EXECUTION_ROLE_NAME=your_selected_ecs_execution_role_name

aws cloudformation create-change-set --stack-name CWAgent-Prometheus-ECS-
${ECS_CLUSTER_NAME}-EC2-${ECS_NETWORK_MODE} \
    --template-body file://cwagent-ecs-prometheus-metric-for-bridge-host.yaml \
    --parameters ParameterKey=ECSClusterName,ParameterValue=$ECS_CLUSTER_NAME \
                 ParameterKey=CreateIAMRoles,ParameterValue=$CREATE_IAM_ROLES \
                 ParameterKey=ECSNetworkMode,ParameterValue=$ECS_NETWORK_MODE \
                 ParameterKey=TaskRoleName,ParameterValue=$ECS_TASK_ROLE_NAME \
                 ParameterKey=ExecutionRoleName,ParameterValue=
$ECS_EXECUTION_ROLE_NAME \
    --capabilities CAPABILITY_NAMED_IAM \
    --region $AWS_REGION \
    --change-set-name nginx-plus-scraping-support
```

5. https://console.aws.amazon.com/cloudformation で AWS CloudFormation コンソール を開きます。
6. 新しく作成されたチェンジセット nginx-plus-scraping-support を確認します。CWAgentConfigSSMParameter リソースに適用された変更が 1 つ表示されます。次のコマンドを入力して、変更セットを実行し、CloudWatch エージェントタスクを再起動します。

```
aws ecs update-service --cluster $ECS_CLUSTER_NAME \
--desired-count 0 \
--service cwagent-prometheus-replica-service-EC2-$ECS_NETWORK_MODE \
--region $AWS_REGION
```

7. 10 秒ほど待ってから、次のコマンドを入力します。

```
aws ecs update-service --cluster $ECS_CLUSTER_NAME \
--desired-count 1 \
--service cwagent-prometheus-replica-service-EC2-$ECS_NETWORK_MODE \
--region $AWS_REGION
```

8. クラスターの Prometheus メトリクス収集を使用して CloudWatch エージェントを初めてインストールする場合は、次のコマンドを入力します。

```
ECS_CLUSTER_NAME=your_cluster_name
AWS_REGION=your_aws_region
ECS_NETWORK_MODE=bridge
CREATE_IAM_ROLES=True
ECS_TASK_ROLE_NAME=your_selected_ecs_task_role_name
ECS_EXECUTION_ROLE_NAME=your_selected_ecs_execution_role_name

aws cloudformation create-stack --stack-name CWAgent-Prometheus-ECS-
${ECS_CLUSTER_NAME}-EC2-${ECS_NETWORK_MODE} \
    --template-body file://cwagent-ecs-prometheus-metric-for-bridge-host.yaml \
    --parameters ParameterKey=ECSClusterName,ParameterValue=$ECS_CLUSTER_NAME \
                 ParameterKey=CreateIAMRoles,ParameterValue=$CREATE_IAM_ROLES \
                 ParameterKey=ECSNetworkMode,ParameterValue=$ECS_NETWORK_MODE \
                 ParameterKey=TaskRoleName,ParameterValue=$ECS_TASK_ROLE_NAME \
                 ParameterKey=ExecutionRoleName,ParameterValue=
$ECS_EXECUTION_ROLE_NAME \
    --capabilities CAPABILITY_NAMED_IAM \
    --region $AWS_REGION
```

NGINX Plus メトリクスとログの表示

これで、収集されている NGINX Plus メトリクスを表示できるようになりました。

サンプル NGINX ワークロードのメトリクスを表示するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. クラスターが実行されているリージョンにおいて、左のナビゲーションペインで [Metrics] (メトリクス) を選択します。ContainerInsights/Prometheus 名前空間を検索して、メトリクスを確認します。
3. CloudWatch Logs イベントを表示するには、ナビゲーションペインで [Log Groups (ロググループ)] を選択します。イベントは、ログストリーム *nginx-plus-prometheus-exporter* のロググループ /aws/containerinsights/***your_cluster_name***/prometheus にあります。

新しい Prometheus スクレイピングターゲットを追加するためのチュートリアル: Amazon ECS の Memcached

このチュートリアルでは、EC2 起動タイプの Amazon ECS クラスター上のサンプル Memcached アプリケーションの Prometheus メトリクスをスクレイプする実践的な紹介を提供しま

す。Memcached Prometheus エクスポーターのターゲットは、ECS タスク定義ベースのサービス検出によって CloudWatch エージェントによって自動検出されます。

Memcached は、汎用分散メモリキャッシュシステムです。多くの場合、データおよびオブジェクトを RAM にキャッシュし、外部データソース (データベースや API など) を読み取る回数を減らすことで、動的なデータベース駆動のウェブサイトを高速化するために使用されます。詳細については、「Memcached とは」を参照してください。

memchached_exporter (Apache License 2.0) は、Prometheus 公式エクスポーターの一つです。デフォルトでは、memcache_exporter は `/metrics.` のポート 0.0.0.0:9150 でサービスを提供します。

このチュートリアルでは、次の 2 つの Docker ハブリポジトリの Docker イメージを使用します。

- Memcached
- prom/memcached-exporter

前提条件

Amazon ECS のサンプル Prometheus ワークロードからメトリクスを収集するには、クラスターで Container Insights を実行している必要があります。Container Insights のインストールの詳細については、「Amazon ECS での Container Insights のセットアップ」を参照してください。

トピック

- Amazon ECS EC2 クラスター環境変数を設定する
- Memcached サンプルワークロードのインストール
- Memcached Prometheus メトリクスをスクレイプするように CloudWatch エージェントを設定する
- Memcached メトリクスの表示

Amazon ECS EC2 クラスター環境変数を設定する

Amazon ECS EC2 クラスター環境変数を設定するには

1. まだの場合は、Amazon ECS CLI をインストールしてください。詳細については、「Amazon ECS CLI のインストール」を参照してください。
2. 新しい Amazon ECS クラスター名とリージョンを設定します。次に例を示します。

```
ECS_CLUSTER_NAME=ecs-ec2-memcached-tutorial
AWS_DEFAULT_REGION=ca-central-1
```

3. (オプション) Memcached サンプルワークロードと CloudWatch エージェントをインストールする EC2 起動タイプの Amazon ECS クラスターがまだない場合は、次のコマンドを入力してクラスターを作成できます。

```
ecs-cli up --capability-iam --size 1 \
--instance-type t3.medium \
--cluster $ECS_CLUSTER_NAME \
--region $AWS_REGION
```

このコマンドの予想される結果は次のとおりです。

```
WARN[0000] You will not be able to SSH into your EC2 instances without a key pair.
INFO[0000] Using recommended Amazon Linux 2 AMI with ECS Agent 1.44.4 and Docker
 version 19.03.6-ce
INFO[0001] Created cluster                               cluster=ecs-ec2-memcached-
tutorial region=ca-central-1
INFO[0002] Waiting for your cluster resources to be created...
INFO[0002] Cloudformation stack status
 stackStatus=CREATE_IN_PROGRESS
INFO[0063] Cloudformation stack status
 stackStatus=CREATE_IN_PROGRESS
INFO[0124] Cloudformation stack status
 stackStatus=CREATE_IN_PROGRESS
VPC created: vpc-xxxxxxxxxxxxxxxxx
Security Group created: sg-xxxxxxxxxxxxxxxxx
Subnet created: subnet-xxxxxxxxxxxxxxxxx
Subnet created: subnet-xxxxxxxxxxxxxxxxx
Cluster creation succeeded.
```

Memcached サンプルワークロードのインストール

Prometheus メトリクスを公開する Memcached サンプルワークロードをインストールするには

1. 次のコマンドを入力して、Memcached AWS CloudFormation テンプレートをダウンロードします。

```
curl -O https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-
insights/latest/ecs-task-definition-templates/deployment-mode/replica-service/
cwagent-prometheus/sample_traffic/memcached/memcached-traffic-sample.yaml
```

2. 次のコマンドを入力して、Memcached 用に作成する IAM ロール名を設定します。

```
MEMCACHED_ECS_TASK_ROLE_NAME=memcached-prometheus-demo-ecs-task-role-name
MEMCACHED_ECS_EXECUTION_ROLE_NAME=memcached-prometheus-demo-ecs-execution-role-name
```

3. 次のコマンドを入力して、Memcached サンプルワークロードをインストールします。このサンプルでは、ワークロードを host ネットワークモードでインストールします。

```
MEMCACHED_ECS_NETWORK_MODE=host

aws cloudformation create-stack --stack-name Memcached-Prometheus-Demo-ECS-
$ECS_CLUSTER_NAME-EC2-$MEMCACHED_ECS_NETWORK_MODE \
    --template-body file://memcached-traffic-sample.yaml \
    --parameters ParameterKey=ECSClusterName,ParameterValue=$ECS_CLUSTER_NAME \
                 ParameterKey=ECSNetworkMode,ParameterValue=
$MEMCACHED_ECS_NETWORK_MODE \
                 ParameterKey=TaskRoleName,ParameterValue=
$MEMCACHED_ECS_TASK_ROLE_NAME \
                 ParameterKey=ExecutionRoleName,ParameterValue=
$MEMCACHED_ECS_EXECUTION_ROLE_NAME \
    --capabilities CAPABILITY_NAMED_IAM \
    --region $AWS_REGION
```

AWS CloudFormation スタックは、次の 4 つのリソースを作成します。

- 1 つの ECS タスクロール
- 1 つの ECS タスク実行ロール
- 1 つの Memcached タスク定義
- 1 つの Memcached サービス

Memcached タスク定義では、次の 2 つのコンテナが定義されています。

- プライマリコンテナは単純な Memcached アプリケーションを実行し、アクセス用にポート 11211 を開きます。

- もう一方のコンテナは Redis エクスポータープロセスを実行して、ポート 9150 の Prometheus メトリクスを公開します。これは、CloudWatch エージェントによって検出され、スクレイプされるコンテナです。

Memcached Prometheus メトリクスをスクレイプするように CloudWatch エージェントを設定する

Memcached Prometheus メトリクスをスクレイプするように CloudWatch エージェントを設定するには

1. 次のコマンドを入力して、cwagent-ecs-prometheus-metric-for-awsvpc.yaml の最新バージョンをダウンロードします。

```
curl -O https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-insights/latest/ecs-task-definition-templates/deployment-mode/replica-service/cwagent-prometheus/cloudformation-quickstart/cwagent-ecs-prometheus-metric-for-awsvpc.yaml
```

2. テキストエディタでファイルを開き、value セクションの resource:CWAgentConfigSSMParameter キーの背後にある完全な CloudWatch エージェント設定を見つけます。

   次に、ecs_service_discovery セクションで、次の設定を task_definition_list セクションに追加します。

```
{
    "sd_job_name": "ecs-memcached",
    "sd_metrics_ports": "9150",
    "sd_task_definition_arn_pattern": ".*:task-definition/memcached-prometheus-demo.*:[0-9]+"
},
```

   metric_declaration セクションの場合、デフォルト設定では Memcached メトリクスを許可しません。Memcached メトリクスを許可するには、次のセクションを追加します。デフォルトのインデントパターンに従ってください。

```
{
  "source_labels": ["container_name"],
  "label_matcher": "memcached-exporter-.*",
  "dimensions": [["ClusterName", "TaskDefinitionFamily"]],
  "metric_selectors": [
```

```
    "^memcached_current_(bytes|items|connections)$",
    "^memcached_items_(reclaimed|evicted)_total$",
    "^memcached_(written|read)_bytes_total$",
    "^memcached_limit_bytes$",
    "^memcached_commands_total$"
  ]
},
{
  "source_labels": ["container_name"],
  "label_matcher": "memcached-exporter-.*",
  "dimensions": [["ClusterName", "TaskDefinitionFamily","status","command"],
 ["ClusterName", "TaskDefinitionFamily","command"]],
  "metric_selectors": [
    "^memcached_commands_total$"
  ]
},
```

3. AWS CloudFormation によって Amazon ECS クラスターに CloudWatch エージェントを既にデプロイしている場合は、次のコマンドを入力して変更セットを作成できます。

```
ECS_NETWORK_MODE=bridge
CREATE_IAM_ROLES=True
ECS_TASK_ROLE_NAME=your_selected_ecs_task_role_name
ECS_EXECUTION_ROLE_NAME=your_selected_ecs_execution_role_name

aws cloudformation create-change-set --stack-name CWAgent-Prometheus-ECS-
${ECS_CLUSTER_NAME}-EC2-${ECS_NETWORK_MODE} \
    --template-body file://cwagent-ecs-prometheus-metric-for-bridge-host.yaml \
    --parameters ParameterKey=ECSClusterName,ParameterValue=$ECS_CLUSTER_NAME \
                 ParameterKey=CreateIAMRoles,ParameterValue=$CREATE_IAM_ROLES \
                 ParameterKey=ECSNetworkMode,ParameterValue=$ECS_NETWORK_MODE \
                 ParameterKey=TaskRoleName,ParameterValue=$ECS_TASK_ROLE_NAME \
                 ParameterKey=ExecutionRoleName,ParameterValue=
$ECS_EXECUTION_ROLE_NAME \
    --capabilities CAPABILITY_NAMED_IAM \
    --region $AWS_REGION \
    --change-set-name memcached-scraping-support
```

4. https://console.aws.amazon.com/cloudformation で AWS CloudFormation コンソール を開きます。
5. 新しく作成された変更セット memcached-scraping-support を確認します。CWAgentConfigSSMParameter リソースに 1 つの変更が適用されていることがわかりま

す。次のコマンドを入力して、変更セットを実行し、CloudWatch エージェントタスクを再起動します。

```
aws ecs update-service --cluster $ECS_CLUSTER_NAME \
--desired-count 0 \
--service cwagent-prometheus-replica-service-EC2-$ECS_NETWORK_MODE \
--region $AWS_REGION
```

6. 10 秒ほど待ってから、次のコマンドを入力します。

```
aws ecs update-service --cluster $ECS_CLUSTER_NAME \
--desired-count 1 \
--service cwagent-prometheus-replica-service-EC2-$ECS_NETWORK_MODE \
--region $AWS_REGION
```

7. クラスターの Prometheus メトリクス収集を使用して CloudWatch エージェントを初めてインストールする場合は、次のコマンドを入力します。

```
ECS_NETWORK_MODEE=bridge
CREATE_IAM_ROLES=True
ECS_TASK_ROLE_NAME=your_selected_ecs_task_role_name
ECS_EXECUTION_ROLE_NAME=your_selected_ecs_execution_role_name

aws cloudformation create-stack --stack-name CWAgent-Prometheus-ECS-
${ECS_CLUSTER_NAME}-EC2-${ECS_NETWORK_MODE} \
    --template-body file://cwagent-ecs-prometheus-metric-for-bridge-host.yaml \
    --parameters ParameterKey=ECSClusterName,ParameterValue=$ECS_CLUSTER_NAME \
                 ParameterKey=CreateIAMRoles,ParameterValue=$CREATE_IAM_ROLES \
                 ParameterKey=ECSNetworkMode,ParameterValue=$ECS_NETWORK_MODE \
                 ParameterKey=TaskRoleName,ParameterValue=$ECS_TASK_ROLE_NAME \
                 ParameterKey=ExecutionRoleName,ParameterValue=
$ECS_EXECUTION_ROLE_NAME \
    --capabilities CAPABILITY_NAMED_IAM \
    --region $AWS_REGION
```

Memcached メトリクスの表示

このチュートリアルでは、次のメトリクスを CloudWatch の ECS/ContainerInsights/Prometheus 名前空間に送信します。CloudWatch コンソールを使用して、その名前空間のメトリクスを表示できます。

メトリクス名	ディメンション
memcached_current_items	ClusterName , TaskDefinitionFamily
memcached_current_connections	ClusterName , TaskDefinitionFamily
memcached_limit_bytes	ClusterName , TaskDefinitionFamily
memcached_current_bytes	ClusterName , TaskDefinitionFamily
memcached_written_bytes_total	ClusterName , TaskDefinitionFamily
memcached_read_bytes_total	ClusterName , TaskDefinitionFamily
memcached_items_evicted_total	ClusterName , TaskDefinitionFamily
memcached_items_reclaimed_total	ClusterName , TaskDefinitionFamily
memcached_commands_total	ClusterName , TaskDefinitionFamily ClusterName 、TaskDefinitionFamily、コマンド ClusterName 、TaskDefinitionFamily、ステータス、コマンド

> **Note**
>
> コマンド ディメンションの値には delete、get、cas、set、decr、touch、incr、または flush を指定できます。
> ステータス ディメンションの値は hit、miss、または badvalです。

Memcached Prometheus メトリクスの CloudWatch ダッシュボードを作成することもできます。

Memcached Prometheus メトリクスのダッシュボードを作成するには

1. 環境変数を作成し、以下の値をデプロイに合わせて置き換えます。

```
DASHBOARD_NAME=your_memcached_cw_dashboard_name
ECS_TASK_DEF_FAMILY=memcached-prometheus-demo-$ECS_CLUSTER_NAME-EC2-
$MEMCACHED_ECS_NETWORK_MOD
```

2. 次のコマンドを入力して、ダッシュボードを作成します。

```
curl https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-
container-insights/latest/ecs-task-definition-templates/deployment-mode/
replica-service/cwagent-prometheus/sample_cloudwatch_dashboards/memcached/
cw_dashboard_memcached.json \
| sed "s/{{YOUR_AWS_REGION}}/$AWS_REGION/g" \
| sed "s/{{YOUR_CLUSTER_NAME}}/$ECS_CLUSTER_NAME/g" \
| sed "s/{{YOUR_TASK_DEF_FAMILY}}/$ECS_TASK_DEF_FAMILY/g" \
| xargs -0 aws cloudwatch put-dashboard --dashboard-name ${DASHBOARD_NAME} --region
 $AWS_REGION --dashboard-body
```

Amazon ECS Fargate の Redis Prometheus メトリクスをスクレイプするためのチュートリアル

このチュートリアルでは、Amazon ECS Fargate クラスター内のサンプル Redis アプリケーションの Prometheus メトリクスをスクレイプする実践的な紹介を提供します。Redis Prometheus エクスポーターターゲットは、コンテナのドッカーラベルに基づいて Prometheus メトリクスサポートを持つ CloudWatch エージェントによって自動検出されます。

Redis (https://redis.io/) は、オープンソース (BSD ライセンス済み)、メモリ内のデータ構造ストアで、データベース、キャッシュ、メッセージブローカーとして使用されます。詳細については、「redis」を参照してください。

redis_exporter (MIT ライセンス済み) は、指定されたポートで Redis プロメテウスメトリクスを公開するために使用されます (デフォルト: 0.0.0.0:9121)。詳細については、「redis_exporter」を参照してください。

このチュートリアルでは、次の 2 つの Docker ハブリポジトリの Docker イメージを使用します。

- redis
- redis_exporter

前提条件

Amazon ECS のサンプル Prometheus ワークロードからメトリクスを収集するには、クラスターで Container Insights を実行している必要があります。Container Insights のインストールの詳細については、「Amazon ECS での Container Insights のセットアップ」を参照してください。

トピック

- Amazon ECS Fargate クラスター環境変数を設定する
- Amazon ECS Fargate クラスターのネットワーク環境変数を設定する
- サンプル Redis ワークロードのインストール
- Redis Prometheus メトリクスをスクレイプするように CloudWatch エージェントを設定する
- Redis メトリクスの表示

Amazon ECS Fargate クラスター環境変数を設定する

Amazon ECS Fargate クラスター環境変数を設定するには

1. まだの場合は、Amazon ECS CLI をインストールしてください。詳細については、「Amazon ECS CLI のインストール」を参照してください。
2. 新しい Amazon ECS クラスター名とリージョンを設定します。次に例を示します。

```
ECS_CLUSTER_NAME=ecs-fargate-redis-tutorial
AWS_DEFAULT_REGION=ca-central-1
```

3. (オプション) サンプル Redis ワークロードと CloudWatch エージェントをインストールする Amazon ECS Fargate クラスターがまだない場合は、次のコマンドを入力してクラスターを作成できます。

```
ecs-cli up --capability-iam \
--cluster $ECS_CLUSTER_NAME \
--launch-type FARGATE \
--region $AWS_DEFAULT_REGION
```

このコマンドの予想される結果は次のとおりです。

```
INFO[0000] Created cluster   cluster=ecs-fargate-redis-tutorial region=ca-central-1
INFO[0001] Waiting for your cluster resources to be created...
INFO[0001] Cloudformation stack status   stackStatus=CREATE_IN_PROGRESS
VPC created: vpc-xxxxxxxxxxxxxxxxx
Subnet created: subnet-xxxxxxxxxxxxxxxxx
Subnet created: subnet-xxxxxxxxxxxxxxxxx
Cluster creation succeeded.
```

Amazon ECS Fargate クラスターのネットワーク環境変数を設定する

Amazon ECS Fargate クラスターのネットワーク環境変数を設定するには

1. Amazon ECS クラスターの VPC とサブネット ID を設定します。前の手順で新しいクラスターを作成した場合は、最後のコマンドの結果にこれらの値が表示されます。それ以外の場合は、Redis で使用する既存のクラスターの ID を使用します。

   ```
   ECS_CLUSTER_VPC=vpc-xxxxxxxxxxxxxxxxx
   ECS_CLUSTER_SUBNET_1=subnet-xxxxxxxxxxxxxxxxx
   ECS_CLUSTER_SUBNET_2=subnet-xxxxxxxxxxxxxxxxx
   ```

2. このチュートリアルでは、Redis アプリケーションと CloudWatch エージェントを Amazon ECS クラスターの VPC のデフォルトセキュリティグループにインストールします。デフォルトのセキュリティグループでは、同じセキュリティグループ内のすべてのネットワーク接続が許可されるため、CloudWatch エージェントは Redis コンテナで公開されている Prometheus メトリクスをスクレイプできます。実際の本番環境では、Redis アプリケーションと CloudWatch エージェント専用のセキュリティグループを作成し、それらにカスタマイズされたアクセス許可を設定することもできます。

   次のコマンドを入力して、デフォルトのセキュリティグループ ID を取得します。

   ```
   aws ec2 describe-security-groups \
   --filters Name=vpc-id,Values=$ECS_CLUSTER_VPC  \
   ```

```
--region $AWS_DEFAULT_REGION
```

次に、次のコマンドを入力して、Fargate クラスターデフォルトセキュリティグループ変数を設定します。*my-default-security-group* を前のコマンドから取得した値に置き換えます。

```
ECS_CLUSTER_SECURITY_GROUP=my-default-security-group
```

サンプル Redis ワークロードのインストール

Prometheus メトリクスを公開するサンプル Redis ワークロードをインストールするには

1. 次のコマンドを入力して、Redis AWS CloudFormation テンプレートをダウンロードします。

```
curl -O https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-
insights/latest/ecs-task-definition-templates/deployment-mode/replica-service/
cwagent-prometheus/sample_traffic/redis/redis-traffic-sample.yaml
```

2. 次のコマンドを入力して、Redis 用に作成する IAM ロール名を設定します。

```
REDIS_ECS_TASK_ROLE_NAME=redis-prometheus-demo-ecs-task-role-name
REDIS_ECS_EXECUTION_ROLE_NAME=redis-prometheus-demo-ecs-execution-role-name
```

3. 次のコマンドを入力して、サンプル Redis ワークロードをインストールします。

```
aws cloudformation create-stack --stack-name Redis-Prometheus-Demo-ECS-
$ECS_CLUSTER_NAME-fargate-awsvpc \
    --template-body file://redis-traffic-sample.yaml \
    --parameters ParameterKey=ECSClusterName,ParameterValue=$ECS_CLUSTER_NAME \
                 ParameterKey=SecurityGroupID,ParameterValue=
$ECS_CLUSTER_SECURITY_GROUP \
                 ParameterKey=SubnetID,ParameterValue=$ECS_CLUSTER_SUBNET_1 \
                 ParameterKey=TaskRoleName,ParameterValue=$REDIS_ECS_TASK_ROLE_NAME
 \
                 ParameterKey=ExecutionRoleName,ParameterValue=
$REDIS_ECS_EXECUTION_ROLE_NAME \
    --capabilities CAPABILITY_NAMED_IAM \
    --region $AWS_DEFAULT_REGION
```

AWS CloudFormation スタックは、次の 4 つのリソースを作成します。

- 1 つの ECS タスクロール
- 1 つの ECS タスク実行ロール
- 1 つの Redis タスク定義
- 1 つの Redis サービス

Redis タスク定義では、2 つのコンテナが定義されています。

- プライマリコンテナは単純な Redis アプリケーションを実行し、アクセス用にポート 6379 を開きます。
- もう一方のコンテナは Redis エクスポータープロセスを実行して、ポート 9121 の Prometheus メトリクスを公開します。これは、CloudWatch エージェントによって検出され、スクレイプされるコンテナです。次のドッカーラベルは、CloudWatch エージェントがそれに基づいてこのコンテナを検出できるように定義されています。

```
ECS_PROMETHEUS_EXPORTER_PORT: 9121
```

Redis Prometheus メトリクスをスクレイプするように CloudWatch エージェントを設定する

Redis Prometheus メトリクスをスクレイプするように CloudWatch エージェントを設定するには

1. 次のコマンドを入力して、cwagent-ecs-prometheus-metric-for-awsvpc.yaml の最新バージョンをダウンロードします。

   ```
   curl -O https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-insights/latest/ecs-task-definition-templates/deployment-mode/replica-service/cwagent-prometheus/cloudformation-quickstart/cwagent-ecs-prometheus-metric-for-awsvpc.yaml
   ```

2. テキストエディタでファイルを開き、value セクションの resource:CWAgentConfigSSMParameter キーの背後にある完全な CloudWatch エージェント設定を見つけます。

   次に、ここに示す ecs_service_discovery セクションでは、docker_label ベースのサービス検出が、Redis ECS タスク定義で定義した Docker ラベルと一致する ECS_PROMETHEUS_EXPORTER_PORT をベースとしたデフォルト設定で有効になっています。したがって、このセクションで変更を加える必要はありません。

```
ecs_service_discovery": {
  "sd_frequency": "1m",
  "sd_result_file": "/tmp/cwagent_ecs_auto_sd.yaml",
*  "docker_label": {
  },*
  ...
```

metric_declaration セクションの場合、デフォルト設定では Redis メトリクスを許可しません。Redis メトリクスを許可するには、次のセクションを追加します。デフォルトのインデントパターンに従ってください。

```
{
  "source_labels": ["container_name"],
  "label_matcher": "^redis-exporter-.*$",
  "dimensions": [["ClusterName","TaskDefinitionFamily"]],
  "metric_selectors": [
    "^redis_net_(in|out)put_bytes_total$",
    "^redis_(expired|evicted)_keys_total$",
    "^redis_keyspace_(hits|misses)_total$",
    "^redis_memory_used_bytes$",
    "^redis_connected_clients$"
  ]
},
{
  "source_labels": ["container_name"],
  "label_matcher": "^redis-exporter-.*$",
  "dimensions": [["ClusterName","TaskDefinitionFamily","cmd"]],
  "metric_selectors": [
    "^redis_commands_total$"
  ]
},
{
  "source_labels": ["container_name"],
  "label_matcher": "^redis-exporter-.*$",
  "dimensions": [["ClusterName","TaskDefinitionFamily","db"]],
  "metric_selectors": [
    "^redis_db_keys$"
  ]
},
```

3. AWS CloudFormation によって Amazon ECS クラスターに CloudWatch エージェントを既にデプロイしている場合は、次のコマンドを入力して変更セットを作成できます。

```
ECS_LAUNCH_TYPE=FARGATE
CREATE_IAM_ROLES=True
ECS_CLUSTER_SUBNET=$ECS_CLUSTER_SUBNET_1
ECS_TASK_ROLE_NAME=your_selected_ecs_task_role_name
ECS_EXECUTION_ROLE_NAME=your_selected_ecs_execution_role_name

aws cloudformation create-change-set --stack-name CWAgent-Prometheus-ECS-
$ECS_CLUSTER_NAME-$ECS_LAUNCH_TYPE-awsvpc \
    --template-body file://cwagent-ecs-prometheus-metric-for-awsvpc.yaml \
    --parameters ParameterKey=ECSClusterName,ParameterValue=$ECS_CLUSTER_NAME \
                 ParameterKey=CreateIAMRoles,ParameterValue=$CREATE_IAM_ROLES \
                 ParameterKey=ECSLaunchType,ParameterValue=$ECS_LAUNCH_TYPE \
                 ParameterKey=SecurityGroupID,ParameterValue=
$ECS_CLUSTER_SECURITY_GROUP \
                 ParameterKey=SubnetID,ParameterValue=$ECS_CLUSTER_SUBNET \
                 ParameterKey=TaskRoleName,ParameterValue=$ECS_TASK_ROLE_NAME \
                 ParameterKey=ExecutionRoleName,ParameterValue=
$ECS_EXECUTION_ROLE_NAME \
    --capabilities CAPABILITY_NAMED_IAM \
    --region ${AWS_DEFAULT_REGION} \
    --change-set-name redis-scraping-support
```

4. https://console.aws.amazon.com/cloudformation で AWS CloudFormation コンソール を開きます。

5. 新しく作成された変更セット `redis-scraping-support` を確認します。`CWAgentConfigSSMParameter` リソースに 1 つの変更が適用されていることがわかります。次のコマンドを入力して、変更セットを実行し、CloudWatch エージェントタスクを再起動します。

```
aws ecs update-service --cluster $ECS_CLUSTER_NAME \
--desired-count 0 \
--service cwagent-prometheus-replica-service-$ECS_LAUNCH_TYPE-awsvpc \
--region ${AWS_DEFAULT_REGION}
```

6. 10 秒ほど待ってから、次のコマンドを入力します。

```
aws ecs update-service --cluster $ECS_CLUSTER_NAME \
--desired-count 1 \
--service cwagent-prometheus-replica-service-$ECS_LAUNCH_TYPE-awsvpc \
```

```
--region ${AWS_DEFAULT_REGION}
```

7. クラスターの Prometheus メトリクス収集を使用して CloudWatch エージェントを初めてインストールする場合は、次のコマンドを入力します。

```
ECS_LAUNCH_TYPE=FARGATE
CREATE_IAM_ROLES=True
ECS_CLUSTER_SUBNET=$ECS_CLUSTER_SUBNET_1
ECS_TASK_ROLE_NAME=your_selected_ecs_task_role_name
ECS_EXECUTION_ROLE_NAME=your_selected_ecs_execution_role_name

aws cloudformation create-stack --stack-name CWAgent-Prometheus-ECS-
$ECS_CLUSTER_NAME-$ECS_LAUNCH_TYPE-awsvpc \
    --template-body file://cwagent-ecs-prometheus-metric-for-awsvpc.yaml \
    --parameters ParameterKey=ECSClusterName,ParameterValue=$ECS_CLUSTER_NAME \
                 ParameterKey=CreateIAMRoles,ParameterValue=$CREATE_IAM_ROLES \
                 ParameterKey=ECSLaunchType,ParameterValue=$ECS_LAUNCH_TYPE \
                 ParameterKey=SecurityGroupID,ParameterValue=
$ECS_CLUSTER_SECURITY_GROUP \
                 ParameterKey=SubnetID,ParameterValue=$ECS_CLUSTER_SUBNET \
                 ParameterKey=TaskRoleName,ParameterValue=$ECS_TASK_ROLE_NAME \
                 ParameterKey=ExecutionRoleName,ParameterValue=
$ECS_EXECUTION_ROLE_NAME \
    --capabilities CAPABILITY_NAMED_IAM \
    --region ${AWS_DEFAULT_REGION}
```

Redis メトリクスの表示

このチュートリアルでは、次のメトリクスを CloudWatch の ECS/ContainerInsights/Prometheus 名前空間に送信します。CloudWatch コンソールを使用して、その名前空間のメトリクスを表示できます。

メトリクス名	ディメンション	
redis_net_input_bytes_total	ClusterName、TaskDefinitionFamily	

メトリクス名	ディメンション	
redis_net_output_bytes_total	ClusterName、TaskDefinitionFamily	
redis_expired_keys_total	ClusterName、TaskDefinitionFamily	
redis_evicted_keys_total	ClusterName、TaskDefinitionFamily	
redis_keyspace_hits_total	ClusterName、TaskDefinitionFamily	
redis_keyspace_misses_total	ClusterName、TaskDefinitionFamily	
redis_memory_used_bytes	ClusterName、TaskDefinitionFamily	
redis_connected_clients	ClusterName、TaskDefinitionFamily	
redis_commands_total	ClusterName ,TaskDefinitionFamily ,cmd	
redis_db_keys	ClusterName ,TaskDefinitionFamily ,db	

> **Note**
>
> cmd ディメンションの値には
> `append`、`client`、`command`、`config`、`dbsize`、`flushall`、`get`、`incr`、`info`、`latency`、または `slowlog` を指定できます。
> db ディメンションの値は `db0` から `db15` に指定できます。

また、Redis Prometheus メトリクスの CloudWatch ダッシュボードを作成することもできます。

Redis Prometheus メトリクスのダッシュボードを作成するには

1. 環境変数を作成し、以下の値をデプロイに合わせて置き換えます。

```
DASHBOARD_NAME=your_cw_dashboard_name
ECS_TASK_DEF_FAMILY=redis-prometheus-demo-$ECS_CLUSTER_NAME-fargate-awsvpc
```

2. 次のコマンドを入力して、ダッシュボードを作成します。

```
curl https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-
insights/latest/k8s-deployment-manifest-templates/deployment-mode/service/cwagent-
prometheus/sample_cloudwatch_dashboards/redis/cw_dashboard_redis.json \
| sed "s/{{YOUR_AWS_REGION}}/${REGION_NAME}/g" \
| sed "s/{{YOUR_CLUSTER_NAME}}/${CLUSTER_NAME}/g" \
| sed "s/{{YOUR_NAMESPACE}}/${NAMESPACE}/g" \
```

Amazon EKS および Kubernetes クラスターでの Prometheus メトリクスコレクションの設定と構成

Amazon EKS または Kubernetes を実行しているクラスターから Prometheus メトリクスを収集するには、CloudWatch エージェントをコレクターとして使用するか、AWS Distro for OpenTelemetry コレクターを使用できます。AWS Distro for OpenTelemetry コレクターの使用については、https://aws-otel.github.io/docs/getting-started/container-insights/eks-prometheus を参照してください。

以下のセクションでは、CloudWatch エージェントを使用して Prometheus メトリクスを収集する方法について説明します。Amazon EKS または Kubernetes を実行しているクラスターに Prometheus モニターリングを使用して CloudWatch エージェントをインストールする方法と、追加のターゲットをスクレイプするようにエージェントを設定する方法について説明します。また、Prometheus モニ

ターリングでのテストに使用するサンプルワークロードを設定するためのオプションのチュートリアルも提供します。

トピック

- Amazon EKS および Kubernetes クラスターに Prometheus メトリクスコレクションを使用して CloudWatch エージェントをインストールする

Amazon EKS および Kubernetes クラスターに Prometheus メトリクスコレクションを使用して CloudWatch エージェントをインストールする

このセクションでは、Amazon EKS または Kubernetes を実行しているクラスターで Prometheus モニターリングを使用して CloudWatch エージェントをセットアップする方法について説明します。これを行うと、エージェントは、そのクラスターで実行されている次のワークロードのメトリクスを自動的にスクレイプし、インポートします。

- AWS App Mesh
- NGINX
- Memcached
- Java/JMX
- HAProxy
- Fluent Bit

また、追加の Prometheus ワークロードとソースをスクレイプしてインポートするようにエージェントを設定することもできます。

以下のステップに従って Prometheus メトリクスコレクション用の CloudWatch エージェントをインストールする前に、Amazon EKS でクラスターが実行されているか、Amazon EC2 インスタンスで Kubernetes クラスターが実行されている必要があります。

VPC セキュリティグループの要件

Prometheus ワークロードのセキュリティグループの受信ルールでは、Prometheus のメトリクスをプライベート IP でスクレイピングするために、CloudWatch エージェントへの Prometheus ポートを開く必要があります。

CloudWatch エージェントのセキュリティグループの出力ルールでは、CloudWatch エージェントがプライベート IP によって Prometheus ワークロードのポートに接続できるようにする必要があります。

トピック

- Amazon EKS および Kubernetes クラスターに Prometheus メトリクスコレクションを使用して CloudWatch エージェントをインストールする
- 追加の Prometheus ソースのスクレイピングと、それらのメトリクスのインポート
- (オプション) Prometheus メトリクスのテストのためにコンテナ化された Amazon EKS サンプルワークロードを設定する

Amazon EKS および Kubernetes クラスターに Prometheus メトリクスコレクションを使用して CloudWatch エージェントをインストールする

このセクションでは、Amazon EKS または Kubernetes を実行しているクラスターで Prometheus モニターリングを使用して CloudWatch エージェントをセットアップする方法について説明します。これを行うと、エージェントは、そのクラスターで実行されている次のワークロードのメトリクスを自動的にスクレイプし、インポートします。

- AWS App Mesh
- NGINX
- Memcached
- Java/JMX
- HAProxy
- Fluent Bit

また、追加の Prometheus ワークロードとソースをスクレイプしてインポートするようにエージェントを設定することもできます。

以下のステップに従って Prometheus メトリクスコレクション用の CloudWatch エージェントをインストールする前に、Amazon EKS でクラスターが実行されているか、Amazon EC2 インスタンスで Kubernetes クラスターが実行されている必要があります。

VPC セキュリティグループの要件

Prometheus ワークロードのセキュリティグループの受信ルールでは、Prometheus のメトリクスをプライベート IP でスクレイピングするために、CloudWatch エージェントへの Prometheus ポートを開く必要があります。

CloudWatch エージェントのセキュリティグループの出力ルールでは、CloudWatch エージェントがプライベート IP によって Prometheus ワークロードのポートに接続できるようにする必要があります。

トピック

- IAM ロールの設定
- Prometheus メトリクスを収集するための CloudWatch エージェントのインストール

IAM ロールの設定

最初のステップでは、クラスターで必要な IAM ロールを設定します。2 つの方法があります。

- サービスアカウントの IAM ロール (サービスロールとも呼ばれます) を設定します。このメソッドは、EC2 起動タイプと Fargate 起動タイプの両方で機能します。
- クラスターに使用される IAM ロールに IAM ポリシーを追加します。これは EC2 起動タイプでのみ機能します。

サービスロール (EC2 起動タイプと Fargate 起動タイプ) を設定する

サービスロールを設定するには、次のコマンドを入力します。*MyCluster* をクラスターの名前に置き換えます。

```
eksctl create iamserviceaccount \
 --name cwagent-prometheus \
--namespace amazon-cloudwatch \
 --cluster MyCluster \
--attach-policy-arn arn:aws:iam::aws:policy/CloudWatchAgentServerPolicy \
--approve \
--override-existing-serviceaccounts
```

クラスターの IAM ロールにポリシーを追加する (EC2 起動タイプのみ)

Prometheus のサポートのためにクラスターで IAM ポリシーを設定するには

1. Amazon EC2 コンソール (https://console.aws.amazon.com/ec2/) を開きます。
2. ナビゲーションペインで、[インスタンス] を選択します。
3. クラスターの IAM ロール名のプレフィックスを検索する必要があります。これを行うには、クラスター内のインスタンスの名前の横にあるチェックボックスをオンにし、[アクション]、[イン

スタンス設定]、[Attach/Replace IAM Role (IAM ロールのアタッチ/置換)] の順に選択します。次に、IAM ロールのプレフィックス (`eksctl-dev303-workshop-nodegroup` など) をコピーします。

4. IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
5. ナビゲーションペインで Roles (ロール) を選択します。
6. 検索ボックスを使用して、この手順で先ほどコピーしたプレフィックスを検索し、そのロールを選択します。
7. [ポリシーのアタッチ] を選択します。
8. 検索ボックスを使用して、[CloudWatchAgentServerPolicy] を検索します。[CloudWatchAgentServerPolicy] の横にあるチェックボックスをオンにして、[Attach policy (ポリシーをアタッチ)] を選択します。

Prometheus メトリクスを収集するための CloudWatch エージェントのインストール

メトリクスを収集するには、CloudWatch エージェントをクラスターにインストールする必要があります。エージェントのインストール方法は、Amazon EKS クラスターと Kubernetes クラスターによって異なります。

Prometheus をサポートする以前のバージョンの CloudWatch エージェントを削除する

Prometheus をサポートするバージョンの CloudWatch エージェントが既にクラスターにインストールされている場合は、次のコマンドを入力してそのバージョンを削除する必要があります。この操作は、Prometheus をサポートする以前のバージョンのエージェントでのみ必要です。Prometheus をサポートせずに Container Insights を有効にする CloudWatch エージェントを削除する必要はありません。

```
kubectl delete deployment cwagent-prometheus -n amazon-cloudwatch
```

EC2 起動タイプの Amazon EKS クラスターへの CloudWatch エージェントのインストール

Prometheus をサポートする CloudWatch エージェントを Amazon EKS クラスターにインストールするには、次の手順に従います。

Amazon EKS クラスターに Prometheus サポートをする CloudWatch エージェントをインストールするには

1. 次のコマンドを入力して、`amazon-cloudwatch` 名前空間がすでに作成されているかどうかを確認します。

```
kubectl get namespace
```

2. 結果に amazon-cloudwatch が表示されない場合は、次のコマンドを入力して作成します。

```
kubectl create namespace amazon-cloudwatch
```

3. デフォルトの設定でエージェントをデプロイし、インストール先の AWS リージョンにデータを送信するには、次のコマンドを入力します。

```
kubectl apply -f https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-
container-insights/latest/k8s-deployment-manifest-templates/deployment-mode/
service/cwagent-prometheus/prometheus-eks.yaml
```

エージェントから代わりに別のリージョンにデータを送信するには、次のステップに従います。

a. 次のコマンドを入力して、エージェントの YAML ファイルをダウンロードします。

```
curl -O https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-
container-insights/latest/k8s-deployment-manifest-templates/deployment-mode/
service/cwagent-prometheus/prometheus-eks.yaml
```

b. テキストエディタでファイルを開き、ファイルの cwagentconfig.json ブロックを検索します。

c. ハイライト表示された行を追加し、必要なリージョンを指定します。

```
cwagentconfig.json: |
    {
      "agent": {
        "region": "us-east-2"
      },
      "logs": { ...
```

d. ファイルを保存し、更新されたファイルを使用してエージェントをデプロイします。

```
kubectl apply -f prometheus-eks.yaml
```

Fargate 起動タイプの Amazon EKS クラスターへの CloudWatch エージェントのインストール

Prometheus をサポートする CloudWatch エージェントを Fargate 起動タイプの Amazon EKS クラスターにインストールするには、次の手順に従います。

Prometheus をサポートする CloudWatch エージェントを Fargate 起動タイプの Amazon EKS クラスターにインストールするには

1. クラスター内で実行できるように、次のコマンドを入力して、CloudWatch エージェントの Fargate プロファイルを作成します。*MyCluster* をクラスターの名前に置き換えます。

   ```
   eksctl create fargateprofile --cluster MyCluster \
   --name amazon-cloudwatch \
   --namespace amazon-cloudwatch
   ```

2. CloudWatch エージェントをインストールするには、次のコマンドを入力します。*MyCluster* をクラスターの名前に置き換えます。この名前は、エージェントによって収集されたログイベントを保存するロググループ名に使用されます。また、エージェントによって収集されたメトリクスのディメンションとしても使用されます。

   region は、メトリクスの送信先となるリージョンの名前に置き換えます。例えば、us-west-1 と指定します。

   ```
   curl https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-insights/latest/k8s-deployment-manifest-templates/deployment-mode/service/cwagent-prometheus/prometheus-eks-fargate.yaml |
   sed "s/{{cluster_name}}/MyCluster/;s/{{region_name}}/region/" |
   kubectl apply -f -
   ```

Kubernetes クラスターへの CloudWatch エージェントのインストール

Kubernetes を実行しているクラスターに Prometheus をサポートする CloudWatch エージェントをインストールするには、次のコマンドを入力します。

```
curl https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-insights/latest/k8s-deployment-manifest-templates/deployment-mode/service/cwagent-prometheus/prometheus-k8s.yaml |
sed "s/{{cluster_name}}/MyCluster/;s/{{region_name}}/region/" |
kubectl apply -f -
```

MyCluster をクラスターの名前に置き換えます。この名前は、エージェントによって収集されたログイベントを保存するロググループ名に使用されます。また、エージェントによって収集されたメトリクスのディメンションとしても使用されます。

region は、メトリクスの送信先となる AWS リージョンの名前に置き換えます。例えば、**us-west-1** と指定します。

エージェントが実行されていることを確認する

Amazon EKS と Kubernetes クラスターの両方で、次のコマンドを入力して、エージェントが実行中であることを確認できます。

```
kubectl get pod -l "app=cwagent-prometheus" -n amazon-cloudwatch
```

結果に Running 状態の 1 つの CloudWatch エージェントポッドが含まれる場合、エージェントは実行中であり、Prometheus メトリクスを収集しています。デフォルトでは、CloudWatch エージェントは、App Mesh、NGINX、Memcached、Java/JMX、および HAProxy のメトリクスを毎分収集します。これらのメトリクスの詳細については、「CloudWatch エージェントにより収集される Prometheus メトリクス」を参照してください。CloudWatch で Prometheus メトリクスを表示する方法については「Prometheus メトリクスの表示 」を参照してください。

他の Prometheus エクスポーターからメトリクスを収集するよう CloudWatch エージェントを設定することもできます。詳細については、「追加の Prometheus ソースのスクレイピングと、それらのメトリクスのインポート」を参照してください。

追加の Prometheus ソースのスクレイピングと、それらのメトリクスのインポート

Prometheus モニターリングを使用した CloudWatch エージェントは、Prometheus メトリクスをスクレイプするために 2 つの設定が必要です。1 つは標準の Prometheus 設定用で、Prometheus ドキュメントの「<scrape_config>」に記載されています。もう 1 つは CloudWatch エージェント設定用です。

Amazon EKS クラスターの場合、設定は `prometheus-eks.yaml` (EC2 起動タイプの場合) または `prometheus-eks-fargate.yaml` (Fargate 起動タイプの場合) で 2 つの設定マップとして定義されます。

- `name: prometheus-config` セクションには、Prometheus のスクレイピング設定が含まれています。
- `name: prometheus-cwagentconfig` セクションには、CloudWatch エージェントの設定が含まれています。このセクションを使用して、CloudWatch で Prometheus メトリクスを収集する方法

を設定できます。例えば、どのメトリクスを CloudWatch にインポートするかを指定し、ディメンションを定義します。

Amazon EC2 インスタンスで実行されている Kubernetes クラスターの場合、設定は `prometheus-k8s.yaml` YAML ファイルで 2 つの設定マップとして定義されます。

- `name: prometheus-config` セクションには、Prometheus のスクレイピング設定が含まれています。
- `name: prometheus-cwagentconfig` セクションには、CloudWatch エージェントの設定が含まれています。

追加の Prometheus メトリクスソースをスクレイプし、それらのメトリクスを CloudWatch にインポートするには、Prometheus スクレイプ設定と CloudWatch エージェント設定の両方を変更し、更新された設定でエージェントを再デプロイします。

VPC セキュリティグループの要件

Prometheus ワークロードのセキュリティグループの受信ルールでは、Prometheus のメトリクスをプライベート IP でスクレイピングするために、CloudWatch エージェントへの Prometheus ポートを開く必要があります。

CloudWatch エージェントのセキュリティグループの出力ルールでは、CloudWatch エージェントがプライベート IP によって Prometheus ワークロードのポートに接続できるようにする必要があります。

Prometheus スクレイプ設定

この CloudWatch エージェントは、Prometheus のドキュメントの「<scrape_config>」に記載されているように、標準の Prometheus スクレイプ設定をサポートしています。このセクションを編集して、このファイルに既に含まれている設定を更新したり、Prometheus スクレイピングターゲットを追加したりできます。デフォルトでは、サンプル設定ファイルに次のグローバル設定行が含まれています。

```
global:
  scrape_interval: 1m
  scrape_timeout: 10s
```

- scrape_interval – ターゲットをスクレイプする頻度を定義します。
- scrape_timeout – スクレイプリクエストがタイムアウトするまでの待機時間を定義します。

また、ジョブレベルでこれらの設定に対して異なる値を定義し、グローバル設定をオーバーライドすることもできます。

Prometheus スクレイピングジョブ

CloudWatch エージェント YAML ファイルには、既にいくつかのデフォルトのスクレイピングジョブが設定されています。例えば、prometheus-eks.yaml では、デフォルトのスクレイピングジョブは、job_name セクションの scrape_configs 行で設定されています。このファイルで、次のデフォルト kubernetes-pod-jmx セクションは、JMX Exporter メトリクスをスクレイピングします。

```
   - job_name: 'kubernetes-pod-jmx'
      sample_limit: 10000
      metrics_path: /metrics
      kubernetes_sd_configs:
      - role: pod
      relabel_configs:
      - source_labels: [__address__]
        action: keep
        regex: '.*:9404$'
      - action: labelmap
        regex: __meta_kubernetes_pod_label_(.+)
      - action: replace
        source_labels:
        - __meta_kubernetes_namespace
        target_label: Namespace
      - source_labels: [__meta_kubernetes_pod_name]
        action: replace
        target_label: pod_name
      - action: replace
        source_labels:
        - __meta_kubernetes_pod_container_name
        target_label: container_name
      - action: replace
        source_labels:
        - __meta_kubernetes_pod_controller_name
        target_label: pod_controller_name
      - action: replace
        source_labels:
        - __meta_kubernetes_pod_controller_kind
        target_label: pod_controller_kind
      - action: replace
        source_labels:
```

```
        - __meta_kubernetes_pod_phase
        target_label: pod_phase
```

これらのデフォルトターゲットはそれぞれスクレイピングされ、メトリクスは埋め込みメトリクス形式を使用してログイベントで CloudWatch に送信されます。詳細については、「ログ内へのメトリクスの埋め込み」を参照してください。

Amazon EKS および Kubernetes クラスターからのログイベントは、CloudWatch Logs の [/aws/containerinsights/***cluster_name***/prometheus] ロググループに保存されます。Amazon ECS クラスターからのログイベントは、[/aws/ecs/containerinsights/***cluster_name***/prometheus] ロググループに保存されます。

各スクレイピングジョブは、このロググループ内の異なるログストリームに含まれています。例えば、Prometheus スクレイピングジョブ `kubernetes-pod-appmesh-envoy` が App Mesh に対して定義されています。Amazon EKS および Kubernetes クラスターのすべての App Mesh Prometheus メトリクスは、[/aws/containerinsights/***cluster_name***>prometheus/kubernetes-pod-appmesh-envoy/] という名前のログストリームに送信されます。

新しいスクレイピングターゲットを追加するには、YAML ファイルの `job_name` セクションに新しい `scrape_configs` セクションを追加し、エージェントを再起動します。このプロセスの例については、「新しい Prometheus スクレイピングターゲットを追加するためのチュートリアル: Prometheus API サーバーメトリクス」を参照してください。

Prometheus の CloudWatch エージェント設定

CloudWatch エージェント設定ファイルには、Prometheus スクレイピング設定の `prometheus` の `metrics_collected` セクションがあります。これには、次の設定オプションが含まれます。

- cluster_name – ログイベントのラベルとして追加されるクラスター名を指定します。このフィールドはオプションです。これを省略すると、エージェントは Amazon EKS または Kubernetes クラスター名を検出できます。
- log_group_name – スクレイプされた Prometheus メトリクスのロググループ名を指定します。このフィールドはオプションです。これを省略すると、CloudWatch では Amazon EKS および Kubernetes クラスターからのログに /aws/containerinsights/***cluster_name***/prometheus が使用されます。
- prometheus_config_path – Prometheus スクレイプ設定ファイルパスを指定します。このフィールドの値が `env:` で始まる場合、Prometheus スクレイプ設定ファイルの内容は、コンテナの環境変数から取得されます。このフィールドは変更しないでください。

- ecs_service_discovery – Amazon ECS Prometheus サービス検出の設定を指定するセクションです。詳細については、「Amazon ECS クラスターでの自動検出の詳細ガイド」を参照してください。

  ecs_service_discovery セクションには、次のフィールドを含めることができます。

  - sd_frequency は、Prometheus エクスポーターを検出する頻度です。数値と単位サフィックスを指定します。例えば、1m の場合は 1 分に 1 回、30s の場合は 30 秒に 1 回です。有効な単位サフィックスは、ns、us、ms、s、m、h です。

    このフィールドはオプションです。デフォルト値は 60 秒 (1 分) です。

  - sd_target_cluster は、自動検出のターゲット Amazon ECS クラスター名です。このフィールドはオプションです。デフォルトは、CloudWatch エージェントがインストールされている Amazon ECS クラスターの名前です。

  - sd_cluster_region は、ターゲット Amazon ECS クラスターのリージョンです。このフィールドはオプションです。デフォルトは、CloudWatch エージェントがインストールされている Amazon ECS クラスターのリージョンです。

  - sd_result_file は、Prometheus ターゲット結果の YAML ファイルのパスです。Prometheus スクレイプ設定は、このファイルを参照します。

  - docker_label は、docker のラベルベースのサービス検出の設定を指定するために使用できるオプションのセクションです。このセクションを省略すると、docker のラベルベースの検出は使用されません。このセクションには、次のフィールドを含めることができます。

    - sd_port_label は、Prometheus メトリクスのコンテナポートを指定するコンテナの docker ラベル名です。デフォルト値は ECS_PROMETHEUS_EXPORTER_PORT です。コンテナにこの docker ラベルがない場合、CloudWatch エージェントはそれをスキップします。

    - sd_metrics_path_label は、Prometheus メトリクスパスを指定するコンテナの docker ラベル名です。デフォルト値は ECS_PROMETHEUS_METRICS_PATH です。コンテナにこの docker ラベルがない場合は、エージェントはデフォルトパス /metrics を想定します。

    - sd_job_name_label は、Prometheus スクレイプジョブ名を指定するコンテナの docker ラベル名です。デフォルト値は job です。コンテナにこの docker ラベルがない場合、CloudWatch エージェントは Prometheus スクレイプ設定でジョブ名を使用します。

  - task_definition_list は、タスク定義ベースのサービス検出の設定を指定するために使用できるオプションのセクションです。このセクションを省略すると、タスク定義ベースの検出は使用されません。このセクションには、次のフィールドを含めることができます。

    - sd_task_definition_arn_pattern は、検出する Amazon ECS タスク定義を指定するために使用するパターンです。これは正規表現です。

  - sd_metrics_ports に、Prometheus メトリクスの containerPort を示します。containerPort はセミコロンで区切ります。
  - sd_container_name_pattern は、Amazon ECS タスクコンテナ名を指定します。これは正規表現です。
  - sd_metrics_path は、Prometheus のメトリクスパスを指定します。これを省略すると、エージェントはデフォルトのパス /metrics を引き受けます。
  - sd_job_name Prometheus スクレイプジョブ名を指定します。このフィールドを省略すると、CloudWatch エージェントは Prometheus スクレイプ設定のジョブ名を使用します。
- metric_declaration – 生成されるメトリクス形式が埋め込まれたログの配列を指定するセクションです。CloudWatch エージェントがインポートする各 Prometheus ソースには、デフォルトで metric_declaration セクションがあります。これらの各セクションには、次のフィールドが含まれています。
  - label_matcher は、 source_labels に表示されているラベルの値をチェックする正規表現です。一致するメトリクスは、CloudWatch に送信される埋め込みメトリクス形式に含めることができます。

    source_labels で複数のラベルを指定する場合は、^ の正規表現に $ や label_matcher 文字を使用しないことをお勧めします。
  - source_labels は、label_matcher 行によってチェックされるラベルの値を指定します。
  - label_separator は、複数の label_matcher が指定されている場合に、source_labels 行で使用するセパレータを指定します。デフォルト: ;。このデフォルトは、次の例の label_matcher 行で使用されています。
  - metric_selectors は、収集され、CloudWatch に送信されるメトリクスを指定する正規表現です。
  - dimensions は、選択した各メトリクスの CloudWatch ディメンションとして使用されるラベルのリストです。

次の metric_declaration の例を参照してください。

```
"metric_declaration": [
  {
     "source_labels":[ "Service", "Namespace"],
     "label_matcher":"(.*node-exporter.*|.*kube-dns.*);kube-system",
     "dimensions":[
        ["Service", "Namespace"]
     ],
```

```
     "metric_selectors":[
        "^coredns_dns_request_type_count_total$"
     ]
  }
]
```

この例では、次の条件が満たされた場合に口グイベントとして送信される埋め込みメトリクス形式セクションを設定します。

- Service の値には node-exporter または kube-dns が含まれます。
- Namespace の値は kube-system です。
- Prometheus メトリクス coredns_dns_request_type_count_total には、Service ラベルおよび Namespace ラベルの両方が含まれます。

送信される口グイベントには、次の強調表示されたセクションが含まれます。

```
{
   "CloudWatchMetrics":[
      {
         "Metrics":[
            {
               "Name":"coredns_dns_request_type_count_total"
            }
         ],
         "Dimensions":[
            [
               "Namespace",
               "Service"
            ]
         ],
         "Namespace":"ContainerInsights/Prometheus"
      }
   ],
   "Namespace":"kube-system",
   "Service":"kube-dns",
   "coredns_dns_request_type_count_total":2562,
   "eks_amazonaws_com_component":"kube-dns",
   "instance":"192.168.61.254:9153",
   "job":"kubernetes-service-endpoints",
   ...
}
```

新しい Prometheus スクレイピングターゲットを追加するためのチュートリアル: Prometheus API サーバーメトリクス

Kubernetes API サーバーは、デフォルトでエンドポイントで Prometheus メトリクスを公開します。Kubernetes API サーバーのスクレイピング設定に関する公式のサンプルは、GitHub で入手できます。

次のチュートリアルでは、次のステップを実行して Kubernetes API サーバーメトリクスを CloudWatch にインポートする方法を示します。

- CloudWatch エージェントの YAML ファイルに、Kubernetes API サーバーの Prometheus スクレイピング設定を追加します。
- CloudWatch エージェントの YAML ファイルに埋め込まれたメトリクス形式のメトリクス定義を設定します。
- (オプション) Kubernetes API サーバーメトリクスの CloudWatch ダッシュボードを作成します。

> **Note**
>
> Kubernetes API サーバーは、ゲージ、カウンター、ヒストグラム、およびサマリーメトリクスを公開します。このリリースの Prometheus メトリクスサポートでは、CloudWatch はゲージ、カウンター、およびサマリータイプのメトリクスのみをインポートします。

CloudWatch で Kubernetes API サーバー Prometheus メトリクスの収集を開始するには

1. 次のコマンドのいずれかを入力して、最新バージョンの `prometheus-eks.yaml`、`prometheus-eks-fargate.yaml`、または `prometheus-k8s.yaml` ファイルをダウンロードします。

   EC2 起動タイプの Amazon EKS クラスターの場合は、次のコマンドを入力します。

   ```
   curl -O https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-insights/latest/k8s-deployment-manifest-templates/deployment-mode/service/cwagent-prometheus/prometheus-eks.yaml
   ```

   Fargate 起動タイプの Amazon EKS クラスターの場合は、次のコマンドを入力します。

```
curl -O https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-insights/latest/k8s-deployment-manifest-templates/deployment-mode/service/cwagent-prometheus/prometheus-eks-fargate.yaml
```

Amazon EC2 インスタンスで実行されている Kubernetes クラスターの場合は、次のコマンドを入力します。

```
curl -O https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-insights/latest/k8s-deployment-manifest-templates/deployment-mode/service/cwagent-prometheus/prometheus-k8s.yaml
```

2. テキストエディタでファイルを開き、prometheus-config セクションを見つけ、そのセクション内に次のセクションを追加します。次に、変更を保存します。

```
    # Scrape config for API servers
    - job_name: 'kubernetes-apiservers'
      kubernetes_sd_configs:
        - role: endpoints
          namespaces:
            names:
              - default
      scheme: https
      tls_config:
        ca_file: /var/run/secrets/kubernetes.io/serviceaccount/ca.crt
        insecure_skip_verify: true
      bearer_token_file: /var/run/secrets/kubernetes.io/serviceaccount/token
      relabel_configs:
      - source_labels: [__meta_kubernetes_service_name,
 __meta_kubernetes_endpoint_port_name]
        action: keep
        regex: kubernetes;https
      - action: replace
        source_labels:
        - __meta_kubernetes_namespace
        target_label: Namespace
      - action: replace
        source_labels:
        - __meta_kubernetes_service_name
        target_label: Service
```

3. テキストエディタで YAML ファイルを開いている間に、cwagentconfig.json セクションを見つけます。次のサブセクションを追加し、変更を保存します。このセクションでは、API サーバーメトリクスを CloudWatch エージェント許可リストに追加します。次の 3 種類の API サーバーメトリクスを許可リストに追加します。

   - etcd オブジェクト数
   - API サーバー登録コントローラーメトリクス
   - API サーバー要求メトリクス

```
{"source_labels": ["job", "resource"],
  "label_matcher": "^kubernetes-apiservers;(services|daemonsets.apps|
deployments.apps|configmaps|endpoints|secrets|serviceaccounts|replicasets.apps)",
  "dimensions": [["ClusterName","Service","resource"]],
  "metric_selectors": [
  "^etcd_object_counts$"
  ]
},
{"source_labels": ["job", "name"],
   "label_matcher": "^kubernetes-apiservers;APIServiceRegistrationController$",
   "dimensions": [["ClusterName","Service","name"]],
   "metric_selectors": [
   "^workqueue_depth$",
   "^workqueue_adds_total$",
   "^workqueue_retries_total$"
  ]
},
{"source_labels": ["job","code"],
  "label_matcher": "^kubernetes-apiservers;2[0-9]{2}$",
  "dimensions": [["ClusterName","Service","code"]],
  "metric_selectors": [
   "^apiserver_request_total$"
  ]
},
{"source_labels": ["job"],
  "label_matcher": "^kubernetes-apiservers",
  "dimensions": [["ClusterName","Service"]],
  "metric_selectors": [
  "^apiserver_request_total$"
  ]
},
```

4. Prometheus をサポートする CloudWatch エージェントが既にクラスタにデプロイされている場合は、次のコマンドを入力してエージェントを削除する必要があります。

```
kubectl delete deployment cwagent-prometheus -n amazon-cloudwatch
```

5. 次のいずれかのコマンドを入力して、更新した設定で CloudWatch エージェントをデプロイします。EC2 起動タイプの Amazon EKS クラスターの場合は、次を入力します:

```
kubectl apply -f prometheus-eks.yaml
```

Fargate 起動タイプの Amazon EKS クラスターの場合は、次のコマンドを入力します。*MyCluster* および *region* を、デプロイに合った値に置き換えます。

```
cat prometheus-eks-fargate.yaml \
| sed "s/{{cluster_name}}/MyCluster/;s/{{region_name}}/region/" \
| kubectl apply -f -
```

Kubernetes クラスターの場合は、次のコマンドを入力します。*MyCluster* および *region* を、デプロイに合った値に置き換えます。

```
cat prometheus-k8s.yaml \
| sed "s/{{cluster_name}}/MyCluster/;s/{{region_name}}/region/" \
| kubectl apply -f -
```

これを行うと、[/aws/containerinsights/***cluster_name***/prometheus] ロググループに [kubernetes-apiservers] という名前の新しいログストリームが表示されます。このログストリームには、次のような埋め込みメトリクス形式の定義を持つログイベントを含める必要があります。

```
{
   "CloudWatchMetrics":[
      {
         "Metrics":[
            {
               "Name":"apiserver_request_total"
            }
         ],
         "Dimensions":[
            [
               "ClusterName",
```

```
                "Service"
            ]
         ],
         "Namespace":"ContainerInsights/Prometheus"
      }
   ],
   "ClusterName":"my-cluster-name",
   "Namespace":"default",
   "Service":"kubernetes",
   "Timestamp":"1592267020339",
   "Version":"0",
   "apiserver_request_count":0,
   "apiserver_request_total":0,
   "code":"0",
   "component":"apiserver",
   "contentType":"application/json",
   "instance":"192.0.2.0:443",
   "job":"kubernetes-apiservers",
   "prom_metric_type":"counter",
   "resource":"pods",
   "scope":"namespace",
   "verb":"WATCH",
   "version":"v1"
}
```

CloudWatch コンソールで [ContainerInsights/Prometheus] 名前空間のメトリクスを表示できます。オプションで、Prometheus Kubernetes API サーバーメトリクスの CloudWatch ダッシュボードを作成することもできます。

(オプション) Kubernetes API サーバーメトリクスのダッシュボードの作成

ダッシュボードに Kubernetes API サーバーメトリクスを表示するには、前のセクションのステップをまず完了して、CloudWatch でこれらのメトリクスの収集を開始する必要があります。

Kubernetes API サーバーメトリクスのダッシュボードを作成するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. 正しい AWS リージョンが選択されていることを確認します。
3. ナビゲーションペインで、ダッシュボードを選択します。
4. [ダッシュボードの作成] を選択します。新しいダッシュボードの名前を入力し、[ダッシュボードの作成] を選択します。

5. [このダッシュボードに追加] で、[キャンセル] を選択します。
6. [アクション]、[ソースの表示/編集] を選択します。
7. JSON ファイル Kubernetes API ダッシュボードソースをダウンロードします。
8. ダウンロードした JSON ファイルをテキストエディタで開き、次の変更を加えます。
   - すべての {{YOUR_CLUSTER_NAME}} 文字列をクラスターの正確な名前に置き換えます。テキストの前後に空白を追加しないようにしてください。
   - すべての {{YOUR_AWS_REGION}} 文字列を、メトリクスを収集するリージョンの名前に置き換えます。例: us-west-2 テキストの前後に空白を追加しないようにしてください。
9. JSON BLOB 全体をコピーし、CloudWatch コンソールのテキストボックスに貼り付けて、ボックスに既に入力されている内容を置き換えます。
10. [更新]、[ダッシュボードの保存] の順に選択します。

(オプション) Prometheus メトリクスのテストのためにコンテナ化された Amazon EKS サンプルワークロードを設定する

CloudWatch Container Insights で Prometheus メトリクスのサポートをテストするには、次のコンテナ化されたワークロードを 1 つ以上設定できます。Prometheus をサポートする CloudWatch エージェントは、これらの各ワークロードからメトリクスを自動的に収集します。デフォルトで収集されるメトリクスを表示する方法については、「CloudWatch エージェントにより収集される Prometheus メトリクス」を参照してください。

これらのワークロードをインストールする前に、次のコマンドを入力して Helm 3.x をインストールする必要があります。

```
brew install helm
```

詳細については、「Helm」を参照してください。

トピック

- Amazon EKS および Kubernetes の AWS App Mesh サンプルワークロードを設定する
- サンプルトラフィックを使用して、Amazon EKS および Kubernetes で NGINX をセットアップする
- メトリクスエクスポーターを使用して、Amazon EKS および Kubernetes で memcached をセットアップする

- Amazon EKS および Kubernetes で Java/JMX サンプルワークロードをセットアップする
- メトリクスエクスポーターを使用して、Amazon EKS および Kubernetes で HAProxy をセットアップする
- 新しい Prometheus スクレイピングターゲットを追加するためのチュートリアル: Amazon EKS クラスターと Kubernetes クラスター

Amazon EKS および Kubernetes の AWS App Mesh サンプルワークロードを設定する

CloudWatch Container Insights での Prometheus サポートでは、 がサポートされていますAWS App Mesh 次のセクションでは、App Mesh の設定方法について説明します。

CloudWatch Container Insights は、App Mesh Envoy アクセスログを収集することもできます。詳細については、「(オプション) App Mesh Envoy アクセスログを有効にする」を参照してください。

トピック

- EC2 起動タイプの Amazon EKS クラスターまたは Kubernetes クラスターで AWS App Mesh サンプルワークロードを設定する
- Fargate 起動タイプの Amazon EKS クラスターで AWS App Mesh サンプルワークロードを設定する

EC2 起動タイプの Amazon EKS クラスターまたは Kubernetes クラスターで AWS App Mesh サンプルワークロードを設定する

EC2 起動タイプの Amazon EKS を実行しているクラスターまたは Kubernetes クラスターで App Mesh を設定する場合は、以下の手順を使用します。

IAM 許可を設定する

Amazon EKS または Kubernetes ノードグループの IAM ロールに AWSAppMeshFullAccess ポリシーを追加する必要があります。Amazon EKS の場合、このノードグループ名は `eksctl-integ-test-eks-prometheus-NodeInstanceRole-ABCDEFHIJKL` のようになります。Kubernetes の場合は、`nodes.integ-test-kops-prometheus.k8s.local` のようになることもあります。

App Mesh のインストール

App Mesh Kubernetes コントローラーをインストールするには、「App Mesh コントローラー」の手順に従います。

サンプルアプリケーションをインストールする

「aws-app-mesh-examples」には、複数の Kubernetes App Mesh チュートリアルが含まれています。このチュートリアルでは、http ルートがヘッダーを使用して着信リクエストを照合する方法を示すサンプルカラーアプリケーションをインストールします。

App Mesh サンプルアプリケーションを使用して Container Insights をテストするには

1. 「https://github.com/aws/aws-app-mesh-examples/tree/main/walkthroughs/howto-k8s-http-headers」の手順に従ってアプリケーションをインストールします。
2. curler ポッドを起動してトラフィックを生成します。

   ```
   kubectl -n default run -it curler --image=tutum/curl /bin/bash
   ```

3. HTTP ヘッダーを変更し、異なるエンドポイントに対して curl を使用します。次のように curl コマンドを複数回実行します。

   ```
   curl -H "color_header: blue" front.howto-k8s-http-headers.svc.cluster.local:8080/;
    echo;

   curl -H "color_header: red" front.howto-k8s-http-headers.svc.cluster.local:8080/;
    echo;

   curl -H "color_header: yellow" front.howto-k8s-http-
   headers.svc.cluster.local:8080/; echo;
   ```

4. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
5. クラスターが実行されている AWS リージョンで、ナビゲーションペインから [Metrics] (メトリクス) を選択します。このメトリクスは、[ContainerInsights/Prometheus] 名前空間にあります。
6. CloudWatch Logs イベントを表示するには、ナビゲーションペインで [Log Groups (ロググループ)] を選択します。イベントは、ロギングストリーム `/aws/containerinsights/your_cluster_name/prometheus` のロググループ `kubernetes-pod-appmesh-envoy` にあります 。

App Mesh テスト環境の削除

App Mesh とサンプルアプリケーションの使用が終了したら、次のコマンドを使用して不要なリソースを削除します。サンプルアプリケーションを削除するには、次のコマンドを入力します。

```
cd aws-app-mesh-examples/walkthroughs/howto-k8s-http-headers/
```

```
kubectl delete -f _output/manifest.yaml
```

App Mesh コントローラーを削除するには、次のコマンドを入力します。

```
helm delete appmesh-controller -n appmesh-system
```

Fargate 起動タイプの Amazon EKS クラスターで AWS App Mesh サンプルワークロードを設定する

Fargate 起動タイプの Amazon EKS を実行しているクラスターで App Mesh を設定する場合は、以下の手順を使用します。

IAM 許可を設定する

IAM アクセス許可を設定するには、次のコマンドを入力します。*MyCluster* をクラスターの名前に置き換えます。

```
eksctl create iamserviceaccount --cluster MyCluster \
 --namespace howto-k8s-fargate \
 --name appmesh-pod \
 --attach-policy-arn arn:aws:iam::aws:policy/AWSAppMeshEnvoyAccess \
 --attach-policy-arn arn:aws:iam::aws:policy/AWSCloudMapDiscoverInstanceAccess \
 --attach-policy-arn arn:aws:iam::aws:policy/AWSXRayDaemonWriteAccess \
 --attach-policy-arn arn:aws:iam::aws:policy/CloudWatchLogsFullAccess \
 --attach-policy-arn arn:aws:iam::aws:policy/AWSAppMeshFullAccess \
 --attach-policy-arn arn:aws:iam::aws:policy/AWSCloudMapFullAccess \
 --override-existing-serviceaccounts \
 --approve
```

App Mesh のインストール

App Mesh Kubernetes コントローラーをインストールするには、「App Mesh コントローラー」の手順に従います。Fargate 起動タイプの Amazon EKS の指示に従ってください。

サンプルアプリケーションをインストールする

「aws-app-mesh-examples」には、複数の Kubernetes App Mesh チュートリアルが含まれています。このチュートリアルでは、Fargate 起動タイプの Amazon EKS クラスターで動作するサンプルカラーアプリケーションをインストールします。

App Mesh サンプルアプリケーションを使用して Container Insights をテストするには

1. 「https://github.com/aws/aws-app-mesh-examples/tree/main/walkthroughs/howto-k8s-fargate」の手順に従ってアプリケーションをインストールします。

   これらの手順では、正しい Fargate プロファイルを使用して新しいクラスターを作成することを前提としています。既に設定済みの Amazon EKS クラスターを使用する場合は、次のコマンドを使用して、このデモンストレーション用にそのクラスターを設定できます。*MyCluster* をクラスターの名前に置き換えます。

   ```
   eksctl create iamserviceaccount --cluster MyCluster \
    --namespace howto-k8s-fargate \
    --name appmesh-pod \
    --attach-policy-arn arn:aws:iam::aws:policy/AWSAppMeshEnvoyAccess \
    --attach-policy-arn arn:aws:iam::aws:policy/AWSCloudMapDiscoverInstanceAccess \
    --attach-policy-arn arn:aws:iam::aws:policy/AWSXRayDaemonWriteAccess \
    --attach-policy-arn arn:aws:iam::aws:policy/CloudWatchLogsFullAccess \
    --attach-policy-arn arn:aws:iam::aws:policy/AWSAppMeshFullAccess \
    --attach-policy-arn arn:aws:iam::aws:policy/AWSCloudMapFullAccess \
    --override-existing-serviceaccounts \
    --approve
   ```

   ```
   eksctl create fargateprofile --cluster MyCluster \
   --namespace howto-k8s-fargate --name howto-k8s-fargate
   ```

2. フロントアプリケーションのデプロイをポート転送します。

   ```
   kubectl -n howto-k8s-fargate port-forward deployment/front 8080:8080
   ```

3. フロントアプリをカールする:

   ```
   while true; do  curl -s http://localhost:8080/color; sleep 0.1; echo ; done
   ```

4. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
5. クラスターが実行されている AWS リージョンで、ナビゲーションペインから [Metrics] (メトリクス) を選択します。このメトリクスは、[ContainerInsights/Prometheus] 名前空間にあります。
6. CloudWatch Logs イベントを表示するには、ナビゲーションペインで [Log Groups (ロググループ)] を選択します。イベントは、ロギングストリーム `/aws/containerinsights/`*your_cluster_name*`/prometheus` のロググループ `kubernetes-pod-appmesh-envoy` にあります 。

App Mesh テスト環境の削除

App Mesh とサンプルアプリケーションの使用が終了したら、次のコマンドを使用して不要なリソースを削除します。サンプルアプリケーションを削除するには、次のコマンドを入力します。

```
cd aws-app-mesh-examples/walkthroughs/howto-k8s-fargate/
kubectl delete -f _output/manifest.yaml
```

App Mesh コントローラーを削除するには、次のコマンドを入力します。

```
helm delete appmesh-controller -n appmesh-system
```

サンプルトラフィックを使用して、Amazon EKS および Kubernetes で NGINX をセットアップする

NGINX は、ロードバランサーやリバースプロキシとしても使用できるウェブサーバーです。Kubernetes が入力に NGINX を使用する方法については、「kubernetes/ingress-nginx」を参照してください。

Container Insights の Prometheus サポートをテストするために、サンプルトラフィックサービス ingress-nginx をインストールするには

1. 次のコマンドを入力して、Helm ingress-nginx リポジトリを追加します。

   ```
   helm repo add ingress-nginx https://kubernetes.github.io/ingress-nginx
   ```

2. 以下のコマンドを入力します。

   ```
   kubectl create namespace nginx-ingress-sample

   helm install my-nginx ingress-nginx/ingress-nginx \
   --namespace nginx-ingress-sample \
   --set controller.metrics.enabled=true \
   --set-string controller.metrics.service.annotations."prometheus\.io/port"="10254" \
   --set-string controller.metrics.service.annotations."prometheus\.io/scrape"="true"
   ```

3. 次のコマンドを入力して、サービスが正常に開始されたかどうかを確認します。

   ```
   kubectl get service -n nginx-ingress-sample
   ```

   このコマンドの出力には、EXTERNAL-IP 列を含む複数の列が表示されます。

4. NGINX 入力コントローラーの行の EXTERNAL-IP 列の値に EXTERNAL-IP 変数を設定します。

```
EXTERNAL_IP=your-nginx-controller-external-ip
```

5. 次のコマンドを入力して、サンプル NGINX トラフィックを開始します。

```
SAMPLE_TRAFFIC_NAMESPACE=nginx-sample-traffic
curl https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-
insights/latest/k8s-deployment-manifest-templates/deployment-mode/service/cwagent-
prometheus/sample_traffic/nginx-traffic/nginx-traffic-sample.yaml |
sed "s/{{external_ip}}/$EXTERNAL_IP/g" |
sed "s/{{namespace}}/$SAMPLE_TRAFFIC_NAMESPACE/g" |
kubectl apply -f -
```

6. 次のコマンドを入力して、3 つのポッドがすべて Running ステータスであることを確認します。

```
kubectl get pod -n $SAMPLE_TRAFFIC_NAMESPACE
```

ポッドが実行中の場合は、ContainerInsights/Prometheus 名前空間にメトリクスが表示されます。

NGINX とサンプルトラフィックアプリケーションをアンインストールするには

1. 次のコマンドを入力して、サンプルトラフィックサービスを削除します。

```
kubectl delete namespace $SAMPLE_TRAFFIC_NAMESPACE
```

2. Helm リリース名で NGINX egress を削除します。

```
helm uninstall my-nginx --namespace nginx-ingress-sample
kubectl delete namespace nginx-ingress-sample
```

メトリクスエクスポーターを使用して、Amazon EKS および Kubernetes で memcached をセットアップする

memcached は、オープンソースのメモリオブジェクトキャッシュシステムです。詳細については、「Memcached とは」を参照してください。

Fargate 起動タイプのクラスターで memcached を実行している場合は、この手順の各ステップを実行する前に Fargate プロファイルを設定する必要があります。プロファイルを設定するには、次のコマンドを入力します。*MyCluster* をクラスターの名前に置き換えます。

```
eksctl create fargateprofile --cluster MyCluster \
--namespace memcached-sample --name memcached-sample
```

メトリクスエクスポーターとともに memcached をインストールして、Container Insights の Prometheus サポートをテストするには

1. 次のコマンドを入力して、リポジトリを追加します。

   ```
   helm repo add bitnami https://charts.bitnami.com/bitnami
   ```

2. 以下のコマンドを入力して、新しい名前空間を作成します。

   ```
   kubectl create namespace memcached-sample
   ```

3. 次のコマンドを入力して、Memcached をインストールします。

   ```
   helm install my-memcached bitnami/memcached --namespace memcached-sample \
   --set metrics.enabled=true \
   --set-string serviceAnnotations.prometheus\\.io/port="9150" \
   --set-string serviceAnnotations.prometheus\\.io/scrape="true"
   ```

4. 次のコマンドを入力して、実行中のサービスの注釈を確認します。

   ```
   kubectl describe service my-memcached-metrics -n memcached-sample
   ```

   次の 2 つの注釈が表示されます。

   ```
   Annotations:   prometheus.io/port: 9150
                  prometheus.io/scrape: true
   ```

memcached をアンインストールするには

- 以下のコマンドを入力します。

```
helm uninstall my-memcached --namespace memcached-sample
kubectl delete namespace memcached-sample
```

Amazon EKS および Kubernetes で Java/JMX サンプルワークロードをセットアップする

JMX Exporter は、Prometheus メトリクスとして JMX mBeans をスクレイプおよび公開できる公式の Prometheus エクスポーターです。詳細については、prometheus/jmx_exporter を参照してください。

Container Insights は、JMX Exporter を使用して、Java 仮想マシン (JVM)、Java、および Tomcat (Catalina) から定義済みの Prometheus メトリクスを収集できます。

デフォルトの Prometheus スクレイピング設定

デフォルトでは、Prometheus をサポートする CloudWatch エージェントは、Amazon EKS または Kubernetes クラスター内の各ポッドの `http://CLUSTER_IP:9404/metrics` から Java/JMX Prometheus メトリクスをスクレイプします。これは、Prometheus `kubernetes_sd_config` の `role: pod` 検出によって行われます。9404 は、Prometheus によって JMX Exporter に割り当てられたデフォルトのポートです。`role: pod` 検出の詳細については、「ポッド」を参照してください。別のポートまたは metrics_path でメトリクスを公開するように JMX Exporter を設定できます。ポートまたはパスを変更する場合は、CloudWatch エージェント設定マップのデフォルトの jmx scrape_config を更新します。次のコマンドを実行して、現在の CloudWatch エージェント Prometheus 設定を取得します。

```
kubectl describe cm prometheus-config -n amazon-cloudwatch
```

変更するフィールドは、次の例で強調表示されているように、`/metrics` および `regex: '.*:9404$'` フィールドです。

```
job_name: 'kubernetes-jmx-pod'
sample_limit: 10000
metrics_path: /metrics
kubernetes_sd_configs:
- role: pod
relabel_configs:
- source_labels: [__address__]
  action: keep
  regex: '.*:9404$'
```

```
- action: replace
  regex: (.+)
  source_labels:
```

その他の Prometheus スクレイピング設定

Kubernetes サービスによって、Java/JMX Prometheus エクスポーターを使用して一連の Pod で実行されているアプリケーションを公開する場合、Prometheus `role: service` の `role: endpoint` 検出または `kubernetes_sd_config` を使用するように切り替えることもできます。これらの検出方法の詳細については、「 サービス」、「 エンドポイント」および「<kubernetes_sd_config>」を参照してください。

これら 2 つのサービス検出モードによって、より多くのメタラベルが提供されるため、CloudWatch メトリクスディメンションを構築するのに便利です。たとえば、`__meta_kubernetes_service_name` ラベルを `Service` ラベルに書き換えてメトリクスのディメンションに含めることができます。CloudWatch メトリクスとそのディメンションのカスタマイズの詳細については、「 Prometheus の CloudWatch エージェント設定」を参照してください。

JMX Exporter を使用した Docker イメージ

次に、Docker イメージを構築します。次のセクションでは、Dockerfile の例を 2 つ示します。

イメージを構築したら、Amazon EKS または Kubernetes にロードし、次のコマンドを実行して Prometheus メトリクスがポート 9404 で `JMX_EXPORTER` によって公開されていることを確認します。*`$JAR_SAMPLE_TRAFFIC_POD`* を実行中のポッド名に置き換え、*`$JAR_SAMPLE_TRAFFIC_NAMESPACE`* をアプリケーション名前空間に置き換えます。

Fargate 起動タイプのクラスターで JMX Exporter を実行している場合は、この手順の各ステップを実行する前に Fargate プロファイルも設定する必要があります。プロファイルを設定するには、次のコマンドを入力します。*`MyCluster`* をクラスターの名前に置き換えます。

```
eksctl create fargateprofile --cluster MyCluster \
--namespace $JAR_SAMPLE_TRAFFIC_NAMESPACE\
 --name $JAR_SAMPLE_TRAFFIC_NAMESPACE
```

```
kubectl exec $JAR_SAMPLE_TRAFFIC_POD -n $JARCAT_SAMPLE_TRAFFIC_NAMESPACE -- curl
 http://localhost:9404
```

例: Prometheus メトリクスを使用した Apache Tomcat Docker イメージ

Apache Tomcat サーバーは、デフォルトで JMX mBeans を公開します。JMX Exporter と Tomcat を統合して、JMX mBeans を Prometheus メトリクスとして公開できます。次の例の Dockerfile は、テスト用イメージを構築するステップを示しています。

```
# From Tomcat 9.0 JDK8 OpenJDK
FROM tomcat:9.0-jdk8-openjdk

RUN mkdir -p /opt/jmx_exporter

COPY ./jmx_prometheus_javaagent-0.12.0.jar /opt/jmx_exporter
COPY ./config.yaml /opt/jmx_exporter
COPY ./setenv.sh /usr/local/tomcat/bin
COPY your web application.war /usr/local/tomcat/webapps/

RUN chmod  o+x /usr/local/tomcat/bin/setenv.sh

ENTRYPOINT ["catalina.sh", "run"]
```

次のリストでは、この Dockerfile の 4 つの COPY 行について説明します。

- https://github.com/prometheus/jmx_exporter から最新の JMX Exporter jar ファイルをダウンロードします。
- config.yaml は JMX Exporter 設定ファイルです。詳細については、https://github.com/prometheus/jmx_exporter#Configuration を参照してください。

  Java と Tomcat の設定ファイルのサンプルを次に示します。

  ```
  lowercaseOutputName: true
  lowercaseOutputLabelNames: true

  rules:
  - pattern: 'java.lang<type=OperatingSystem><>(FreePhysicalMemorySize|
  TotalPhysicalMemorySize|FreeSwapSpaceSize|TotalSwapSpaceSize|SystemCpuLoad|
  ProcessCpuLoad|OpenFileDescriptorCount|AvailableProcessors)'
    name: java_lang_OperatingSystem_$1
    type: GAUGE

  - pattern: 'java.lang<type=Threading><>(TotalStartedThreadCount|ThreadCount)'
    name: java_lang_threading_$1
    type: GAUGE
  ```

```
- pattern: 'Catalina<type=GlobalRequestProcessor, name=\"(\w+-\w+)-(\d+)\"><>(\w+)'
  name: catalina_globalrequestprocessor_$3_total
  labels:
    port: "$2"
    protocol: "$1"
  help: Catalina global $3
  type: COUNTER

- pattern: 'Catalina<j2eeType=Servlet, WebModule=//([-a-zA-Z0-9+&@#/%?=~_|!:.,;]*[-
a-zA-Z0-9+&@#/%=~_|]), name=([-a-zA-Z0-9+/$%~_-|!.]*), J2EEApplication=none,
 J2EEServer=none><>(requestCount|maxTime|processingTime|errorCount)'
  name: catalina_servlet_$3_total
  labels:
    module: "$1"
    servlet: "$2"
  help: Catalina servlet $3 total
  type: COUNTER

- pattern: 'Catalina<type=ThreadPool, name="(\w+-\w+)-(\d+)"><>(currentThreadCount|
currentThreadsBusy|keepAliveCount|pollerThreadCount|connectionCount)'
  name: catalina_threadpool_$3
  labels:
    port: "$2"
    protocol: "$1"
  help: Catalina threadpool $3
  type: GAUGE

- pattern: 'Catalina<type=Manager, host=([-a-zA-Z0-9+&@#/%?=~_|!:.,;]*[-a-zA-
Z0-9+&@#/%=~_|]), context=([-a-zA-Z0-9+/$%~_-|!.]*)><>(processingTime|sessionCounter|
rejectedSessions|expiredSessions)'
  name: catalina_session_$3_total
  labels:
    context: "$2"
    host: "$1"
  help: Catalina session $3 total
  type: COUNTER

- pattern: ".*"
```

- `setenv.sh` は、Tomcat とともに JMX Exporter を起動し、ローカルホストのポート 9404 で Prometheus メトリクスを公開する Tomcat 起動スクリプトです。また、`config.yaml` ファイルパスを JMX Exporter に提供します。

```
$ cat setenv.sh
export JAVA_OPTS="-javaagent:/opt/jmx_exporter/
jmx_prometheus_javaagent-0.12.0.jar=9404:/opt/jmx_exporter/config.yaml $JAVA_OPTS"
```

- ウェブアプリケーション.war は、Tomcat によってロードされるウェブアプリケーション war ファイルです。

この設定で Docker イメージを構築し、イメージリポジトリにアップロードします。

例: Prometheus メトリクスを使用した Java Jar アプリケーション Docker イメージ

次の例の Dockerfile は、テスト用イメージを構築するステップを示しています。

```
# Alpine Linux with OpenJDK JRE
FROM openjdk:8-jre-alpine

RUN mkdir -p /opt/jmx_exporter

COPY ./jmx_prometheus_javaagent-0.12.0.jar /opt/jmx_exporter
COPY ./SampleJavaApplication-1.0-SNAPSHOT.jar /opt/jmx_exporter
COPY ./start_exporter_example.sh /opt/jmx_exporter
COPY ./config.yaml /opt/jmx_exporter

RUN chmod -R o+x /opt/jmx_exporter
RUN apk add curl

ENTRYPOINT exec /opt/jmx_exporter/start_exporter_example.sh
```

次のリストでは、この Dockerfile の 4 つの COPY 行について説明します。

- https://github.com/prometheus/jmx_exporter から最新の JMX Exporter jar ファイルをダウンロードします。
- config.yaml は JMX Exporter 設定ファイルです。詳細については、https://github.com/prometheus/jmx_exporter#Configuration を参照してください。

  Java と Tomcat の設定ファイルのサンプルを次に示します。

```
lowercaseOutputName: true
lowercaseOutputLabelNames: true
```

```
rules:
- pattern: 'java.lang<type=OperatingSystem><>(FreePhysicalMemorySize|
TotalPhysicalMemorySize|FreeSwapSpaceSize|TotalSwapSpaceSize|SystemCpuLoad|
ProcessCpuLoad|OpenFileDescriptorCount|AvailableProcessors)'
  name: java_lang_OperatingSystem_$1
  type: GAUGE

- pattern: 'java.lang<type=Threading><>(TotalStartedThreadCount|ThreadCount)'
  name: java_lang_threading_$1
  type: GAUGE

- pattern: 'Catalina<type=GlobalRequestProcessor, name=\"(\w+-\w+)-(\d+)\"><>(\w+)'
  name: catalina_globalrequestprocessor_$3_total
  labels:
    port: "$2"
    protocol: "$1"
  help: Catalina global $3
  type: COUNTER

- pattern: 'Catalina<j2eeType=Servlet, WebModule=//([-a-zA-Z0-9+&@#/%?=~_|!:.,;]*[-
a-zA-Z0-9+&@#/%=~_|]), name=([-a-zA-Z0-9+/$%~_-|!.]*), J2EEApplication=none,
 J2EEServer=none><>(requestCount|maxTime|processingTime|errorCount)'
  name: catalina_servlet_$3_total
  labels:
    module: "$1"
    servlet: "$2"
  help: Catalina servlet $3 total
  type: COUNTER

- pattern: 'Catalina<type=ThreadPool, name="(\w+-\w+)-(\d+)"><>(currentThreadCount|
currentThreadsBusy|keepAliveCount|pollerThreadCount|connectionCount)'
  name: catalina_threadpool_$3
  labels:
    port: "$2"
    protocol: "$1"
  help: Catalina threadpool $3
  type: GAUGE

- pattern: 'Catalina<type=Manager, host=([-a-zA-Z0-9+&@#/%?=~_|!:.,;]*[-a-zA-
Z0-9+&@#/%=~_|]), context=([-a-zA-Z0-9+/$%~_-|!.]*)><>(processingTime|sessionCounter|
rejectedSessions|expiredSessions)'
  name: catalina_session_$3_total
  labels:
    context: "$2"
```

```
    host: "$1"
  help: Catalina session $3 total
  type: COUNTER

- pattern: ".*"
```

- `start_exporter_example.sh` は、エクスポートされた Prometheus メトリクスを使用して JAR アプリケーションを起動するスクリプトです。また、`config.yaml` ファイルパスを JMX Exporter に提供します。

```
$ cat start_exporter_example.sh
java -javaagent:/opt/jmx_exporter/jmx_prometheus_javaagent-0.12.0.jar=9404:/
opt/jmx_exporter/config.yaml -cp  /opt/jmx_exporter/SampleJavaApplication-1.0-
SNAPSHOT.jar com.gubupt.sample.app.App
```

- SampleJavaApplication-1.0-SNAPSHOT.jar は、Java アプリケーションのサンプル jar ファイルです。このファイルを、モニターリングする Java アプリケーションに置き換えます。

この設定で Docker イメージを構築し、イメージリポジトリにアップロードします。

メトリクスエクスポーターを使用して、Amazon EKS および Kubernetes で HAProxy をセットアップする

HAProxy は、オープンソースのプロキシアプリケーションです。詳細については、「HAProxy」を参照してください。

Fargate 起動タイプのクラスターで HAProxy を実行している場合は、この手順の各ステップを実行する前に Fargate プロファイルを設定する必要があります。プロファイルを設定するには、次のコマンドを入力します。*MyCluster* をクラスターの名前に置き換えます。

```
eksctl create fargateprofile --cluster MyCluster \
--namespace haproxy-ingress-sample --name haproxy-ingress-sample
```

メトリクスエクスポーターとともに HAProxy をインストールして、Container Insights の Prometheus サポートをテストするには

1. 次のコマンドを入力して、Helm インキュベーターレポジトリを追加します。

   ```
   helm repo add haproxy-ingress https://haproxy-ingress.github.io/charts
   ```

2. 以下のコマンドを入力して、新しい名前空間を作成します。

```
kubectl create namespace haproxy-ingress-sample
```

3. 次のコマンドを入力して、HAProxy をインストールします。

```
helm install haproxy haproxy-ingress/haproxy-ingress \
--namespace haproxy-ingress-sample \
--set defaultBackend.enabled=true \
--set controller.stats.enabled=true \
--set controller.metrics.enabled=true \
--set-string controller.metrics.service.annotations."prometheus\.io/port"="9101" \
--set-string controller.metrics.service.annotations."prometheus\.io/scrape"="true"
```

4. 次のコマンドを入力して、サービスの注釈を確認します。

```
kubectl describe service haproxy-haproxy-ingress-metrics -n haproxy-ingress-sample
```

次の注釈が表示されます。

```
Annotations:   prometheus.io/port: 9101
               prometheus.io/scrape: true
```

HAProxy をアンインストールするには

- 以下のコマンドを入力します。

```
helm uninstall haproxy --namespace haproxy-ingress-sample
kubectl delete namespace haproxy-ingress-sample
```

新しい Prometheus スクレイピングターゲットを追加するためのチュートリアル: Amazon EKS クラスターと Kubernetes クラスター

このチュートリアルでは、Amazon EKS および Kubernetes のサンプル Redis アプリケーションの Prometheus メトリクスをスクレイプするための実践的な紹介を紹介します。Redis (https://redis.io/) は、オープンソース (BSD ライセンス済み)、メモリ内のデータ構造ストアで、データベース、キャッシュ、メッセージブローカーとして使用されます。詳細については、「redis」を参照してください。

redis_exporter (MIT ライセンス済み) は、指定されたポートで Redis プロメテウスメトリクスを公開するために使用されます (デフォルト: 0.0.0.0:9121)。詳細については、「redis_exporter」を参照してください。

このチュートリアルでは、次の 2 つの Docker ハブリポジトリの Docker イメージを使用します。

- redis
- redis_exporter

Prometheus メトリクスを公開するサンプル Redis ワークロードをインストールするには

1. サンプル Redis ワークロードの名前空間を設定します。

```
REDIS_NAMESPACE=redis-sample
```

2. Fargate 起動タイプのクラスターで Redis を実行している場合は、Fargate プロファイルを設定する必要があります。プロファイルを設定するには、次のコマンドを入力します。*MyCluster* をクラスターの名前に置き換えます。

```
eksctl create fargateprofile --cluster MyCluster \
--namespace $REDIS_NAMESPACE --name $REDIS_NAMESPACE
```

3. 次のコマンドを入力して、サンプル Redis ワークロードをインストールします。

```
curl https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-
insights/latest/k8s-deployment-manifest-templates/deployment-mode/service/cwagent-
prometheus/sample_traffic/redis/redis-traffic-sample.yaml \
| sed "s/{{namespace}}/$REDIS_NAMESPACE/g" \
| kubectl apply -f -
```

4. インストールには、ポート 9121 の Redis Prometheus メトリクスを公開する `my-redis-metrics` という名前のサービスが含まれています。サービスの詳細を取得するには、次のコマンドを入力します。

```
kubectl describe service/my-redis-metrics  -n $REDIS_NAMESPACE
```

結果の `Annotations` セクションには、CloudWatch エージェントの Prometheus スクレイプ設定に一致する 2 つの注釈が表示されます。これにより、ワークロードを自動検出できます。

```
prometheus.io/port: 9121
```

```
prometheus.io/scrape: true
```

関連する Prometheus スクレイプの設定は、`- job_name: kubernetes-service-endpoints` または `kubernetes-eks.yaml` の `kubernetes-k8s.yaml` のセクションに記載されています。

CloudWatch で Redis Prometheus メトリクスの収集を開始するには

1. 次のコマンドのいずれかを入力して、最新バージョンの `kubernetes-eks.yaml` または `kubernetes-k8s.yaml` ファイルをダウンロードします。EC2 起動タイプの Amazon EKS クラスターの場合は、次のコマンドを入力します。

   ```
   curl -O https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-insights/latest/k8s-deployment-manifest-templates/deployment-mode/service/cwagent-prometheus/prometheus-eks.yaml
   ```

   Fargate 起動タイプの Amazon EKS クラスターの場合は、次のコマンドを入力します。

   ```
   curl -O https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-insights/latest/k8s-deployment-manifest-templates/deployment-mode/service/cwagent-prometheus/prometheus-eks-fargate.yaml
   ```

   Amazon EC2 インスタンスで実行されている Kubernetes クラスターの場合は、このコマンドを入力します。

   ```
   curl -O https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-insights/latest/k8s-deployment-manifest-templates/deployment-mode/service/cwagent-prometheus/prometheus-k8s.yaml
   ```

2. テキストエディタでファイルを開き、`cwagentconfig.json` セクションを見つけます。次のサブセクションを追加し、変更を保存します。インデントが既存のパターンに従っていることを確認してください。

   ```
   {
     "source_labels": ["pod_name"],
     "label_matcher": "^redis-instance$",
     "dimensions": [["Namespace","ClusterName"]],
     "metric_selectors": [
       "^redis_net_(in|out)put_bytes_total$",
   ```

```
    "^redis_(expired|evicted)_keys_total$",
    "^redis_keyspace_(hits|misses)_total$",
    "^redis_memory_used_bytes$",
    "^redis_connected_clients$"
  ]
},
{
  "source_labels": ["pod_name"],
  "label_matcher": "^redis-instance$",
  "dimensions": [["Namespace","ClusterName","cmd"]],
  "metric_selectors": [
    "^redis_commands_total$"
  ]
},
{
  "source_labels": ["pod_name"],
  "label_matcher": "^redis-instance$",
  "dimensions": [["Namespace","ClusterName","db"]],
  "metric_selectors": [
    "^redis_db_keys$"
  ]
},
```

追加したセクションでは、Redis メトリクスを CloudWatch エージェントの許可リストに配置します。これらのメトリクスのリストについては、次のセクションを参照してください。

3. Prometheus をサポートする CloudWatch エージェントが既にこのクラスターにデプロイされている場合は、次のコマンドを入力してエージェントを削除する必要があります。

```
kubectl delete deployment cwagent-prometheus -n amazon-cloudwatch
```

4. 次のいずれかのコマンドを入力して、更新した設定で CloudWatch エージェントをデプロイします。*MyCluster* と *#####*を設定に合わせて置き換えます。

   EC2 起動タイプの Amazon EKS クラスターの場合は、次のコマンドを入力します。

```
kubectl apply -f prometheus-eks.yaml
```

   Fargate 起動タイプの Amazon EKS クラスターの場合は、次のコマンドを入力します。

```
cat prometheus-eks-fargate.yaml \
| sed "s/{{cluster_name}}/MyCluster/;s/{{region_name}}/region/" \
```

```
| kubectl apply -f -
```

Kubernetes クラスターの場合は、このコマンドを入力します。

```
cat prometheus-k8s.yaml \
| sed "s/{{cluster_name}}/MyCluster/;s/{{region_name}}/region/" \
| kubectl apply -f -
```

Redis Prometheus メトリクスの表示

このチュートリアルでは、次のメトリクスを CloudWatch の ContainerInsights/Prometheus 名前空間に送信します。CloudWatch コンソールを使用して、その名前空間のメトリクスを表示できます。

メトリクス名	ディメンション	
redis_net_input_bytes_total	ClusterName、Namespace	
redis_net_output_bytes_total	ClusterName、Namespace	
redis_expired_keys_total	ClusterName、Namespace	
redis_evicted_keys_total	ClusterName、Namespace	
redis_keyspace_hits_total	ClusterName、Namespace	
redis_keyspace_misses_total	ClusterName、Namespace	

メトリクス名	ディメンション	
redis_memory_used_bytes	ClusterName、Namespace	
redis_connected_clients	ClusterName、Namespace	
redis_commands_total	ClusterName、Namespace 、cmd	
redis_db_keys	ClusterName、Namespace 、db	

> **Note**
>
> cmd ディメンションの値には append、client、command、config、dbsize、flushall、get、incr、info、latency、または slowlog を指定できます。
> db ディメンションの値は db0 から db15 に指定できます。

また、Redis Prometheus メトリクスの CloudWatch ダッシュボードを作成することもできます。

Redis Prometheus メトリクスのダッシュボードを作成するには

1. 環境変数を作成し、以下の値をデプロイに合わせて置き換えます。

```
DASHBOARD_NAME=your_cw_dashboard_name
REGION_NAME=your_metric_region_such_as_us-east-1
CLUSTER_NAME=your_k8s_cluster_name_here
NAMESPACE=your_redis_service_namespace_here
```

2. 次のコマンドを入力して、ダッシュボードを作成します。

```
curl https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-
insights/latest/k8s-deployment-manifest-templates/deployment-mode/service/cwagent-
prometheus/sample_cloudwatch_dashboards/redis/cw_dashboard_redis.json \
| sed "s/{{YOUR_AWS_REGION}}/${REGION_NAME}/g" \
| sed "s/{{YOUR_CLUSTER_NAME}}/${CLUSTER_NAME}/g" \
```

```
| sed "s/{{YOUR_NAMESPACE}}/${NAMESPACE}/g" \
```

CloudWatch エージェントによる Prometheus メトリクスタイプの変換

Prometheus クライアントライブラリには、次の 4 つのコアメトリクスタイプがあります。

- Counter
- Gauge
- 概要
- Histogram

CloudWatch エージェントは、カウンター、ゲージ、サマリーメトリクスタイプをサポートします。ヒストグラムメトリクスのサポートは、今後のリリースで予定されています。

サポートされていないヒストグラムメトリクスタイプの Prometheus メトリクスは、CloudWatch エージェントによって除外されます。詳細については、「削除された Prometheus メトリクスの口グ記録」を参照してください。

ゲージメトリクス

プロメテウスゲージメトリクス (Prometheus gauge metric) とは、任意に上下できる単一の数値を表すメトリクスです。CloudWatch エージェントはゲージメトリクスをスクレイプし、これらの値を直接送信します。

カウンターメトリクス

Prometheus カウンターメトリクスは、値を増加またはゼロにリセットすることができる単一の単調増加カウンターを表す累積メトリクスです。CloudWatch エージェントは、前回のスクレイプからデルタを計算し、ログイベントのメトリクス値としてデルタ値を送信します。したがって、CloudWatch エージェントは 2 回目のスクレイプから 1 つのログイベントを生成し始め、後続のスクラップがあれば続行します。

サマリーメトリクス

Prometheus サマリーメトリクスは、複数のデータポイントによって表される複雑なメトリクスタイプです。これは、観測値の合計数とすべての観測値の合計を提供します。スライドのタイムウィンドウで設定可能な分位数を計算します。

サマリーメトリクスの合計とカウントは累積されますが、分位数は累積されません。次の例は、分位数の分散を示しています。

```
# TYPE go_gc_duration_seconds summary
go_gc_duration_seconds{quantile="0"} 7.123e-06
go_gc_duration_seconds{quantile="0.25"} 9.204e-06
go_gc_duration_seconds{quantile="0.5"} 1.1065e-05
go_gc_duration_seconds{quantile="0.75"} 2.8731e-05
go_gc_duration_seconds{quantile="1"} 0.003841496
go_gc_duration_seconds_sum 0.37630427
go_gc_duration_seconds_count 9774
```

前のセクションで説明したように、CloudWatch エージェントはカウンターメトリクスを処理するのと同じ方法で、サマリーメトリクスの合計とカウントを処理します。CloudWatch エージェントは、最初に報告された分位数の値を保持します。

CloudWatch エージェントにより収集される Prometheus メトリクス

Prometheus をサポートする CloudWatch エージェントは、複数のサービスとワークロードからメトリクスを自動的に収集します。デフォルトで収集されるメトリクスは、次のセクションに表示されます。これらのサービスから収集するメトリクスを増やし、他のアプリケーションやサービスから Prometheus メトリクスを収集するようにエージェントを設定することもできます。追加メトリクスの収集の詳細については、「Prometheus の CloudWatch エージェント設定」を参照してください。

Amazon EKS および Kubernetes クラスターから収集された Prometheus メトリクスは、[ContainerInsights/Prometheus] 名前空間にあります。Amazon ECS クラスターから収集された Prometheus メトリクスは、 [ECS/ContainerInsights/Prometheus] 名前空間にあります。

トピック

- App Mesh の Prometheus メトリクス
- NGINX の Prometheus メトリクス
- memcached の Prometheus メトリクス
- Java/JMX の Prometheus メトリクス
- HAProxy の Prometheus メトリクス

App Mesh の Prometheus メトリクス

次のメトリクスは App Mesh から自動的に収集されます。

CloudWatch Container Insights は、App Mesh Envoy アクセスログを収集することもできます。詳細については、「(オプション) App Mesh Envoy アクセスログを有効にする」を参照してください。

Amazon EKS クラスターおよび Kubernetes クラスターでの App Mesh の Prometheus メトリクス

メトリクス名	ディメンション	
envoy_http_downstream_rq_total	ClusterName、Namespace	
envoy_http_downstream_rq_xx	ClusterName、Namespace ClusterName、Namespace 、envoy_http_conn_manager_prefix、envoy_response_code_class	
envoy_cluster_upstream_cx_rx_bytes_total	ClusterName、Namespace	
envoy_cluster_upstream_cx_tx_bytes_total	ClusterName、Namespace	
envoy_cluster_membership_healthy	ClusterName、Namespace	
envoy_cluster_membership_total	ClusterName、Namespace	
envoy_server_memory_heap_size	ClusterName、Namespace	

メトリクス名	ディメンション	
envoy_server_memory_allocated	ClusterName、Namespace	
envoy_cluster_upstream_cx_connect_timeout	ClusterName、Namespace	
envoy_cluster_upstream_rq_pending_failure_eject	ClusterName、Namespace	
envoy_cluster_upstream_rq_pending_overflow	ClusterName、Namespace	
envoy_cluster_upstream_rq_timeout	ClusterName、Namespace	
envoy_cluster_upstream_rq_try_per_timeout	ClusterName、Namespace	
envoy_cluster_upstream_rq_rx_reset	ClusterName、Namespace	

メトリクス名	ディメンション	
envoy_cluster_upstream_cx_destroy_local_with_active_rq	ClusterName、Namespace	
envoy_cluster_upstream_cx_destroy_remote_active_rq	ClusterName、Namespace	
envoy_cluster_upstream_rq_maintenance_mode	ClusterName、Namespace	
envoy_cluster_upstream_flow_control_paused_reading_total	ClusterName、Namespace	
envoy_cluster_upstream_flow_control_resumed_reading_total	ClusterName、Namespace	

メトリクス名	ディメンション	
envoy_cluster_upstream_flow_control_backed_up_total	ClusterName、Namespace	
envoy_cluster_upstream_flow_control_drained_total	ClusterName、Namespace	
envoy_cluster_upstream_rq_retry	ClusterName、Namespace	
envoy_cluster_upstream_rq_retry_success	ClusterName、Namespace	
envoy_cluster_upstream_rq_retry_overflow	ClusterName、Namespace	
envoy_server_live	ClusterName、Namespace	
envoy_server_uptime	ClusterName、Namespace	

Amazon ECS クラスターでの App Mesh の Prometheus メトリクス

メトリクス名	ディメンション	
envoy_http_downstream_rq_total	ClusterName、TaskDefinitionFamily	
envoy_http_downstream_rq_xx	ClusterName、TaskDefinitionFamily	
envoy_cluster_upstream_cx_rx_bytes_total	ClusterName、TaskDefinitionFamily	
envoy_cluster_upstream_cx_tx_bytes_total	ClusterName、TaskDefinitionFamily	
envoy_cluster_membership_healthy	ClusterName、TaskDefinitionFamily	
envoy_cluster_membership_total	ClusterName、TaskDefinitionFamily	
envoy_server_memory_heap_size	ClusterName、TaskDefinitionFamily	
envoy_server_memory_allocated	ClusterName、TaskDefinitionFamily	
envoy_cluster_upst	ClusterName、TaskDefinitionFamily	

メトリクス名	ディメンション	
ream_cx_connect_timeout		
envoy_cluster_upstream_rq_pending_failure_eject	ClusterName、TaskDefinitionFamily	
envoy_cluster_upstream_rq_pending_overflow	ClusterName、TaskDefinitionFamily	
envoy_cluster_upstream_rq_timeout	ClusterName、TaskDefinitionFamily	
envoy_cluster_upstream_rq_try_per_timeout	ClusterName、TaskDefinitionFamily	
envoy_cluster_upstream_rq_rx_reset	ClusterName、TaskDefinitionFamily	
envoy_cluster_upstream_cx_destroy_local_with_active_rq	ClusterName、TaskDefinitionFamily	

メトリクス名	ディメンション	
envoy_cluster_upstream_cx_destroy_remote_active_rq	ClusterName、TaskDefinitionFamily	
envoy_cluster_upstream_rq_maintenance_mode	ClusterName、TaskDefinitionFamily	
envoy_cluster_upstream_flow_control_paused_reading_total	ClusterName、TaskDefinitionFamily	
envoy_cluster_upstream_flow_control_resumed_reading_total	ClusterName、TaskDefinitionFamily	
envoy_cluster_upstream_flow_control_backed_up_total	ClusterName、TaskDefinitionFamily	

メトリクス名	ディメンション
envoy_cluster_upstream_flow_control_drained_total	ClusterName、TaskDefinitionFamily
envoy_cluster_upstream_rq_retry	ClusterName、TaskDefinitionFamily
envoy_cluster_upstream_rq_retry_success	ClusterName、TaskDefinitionFamily
envoy_cluster_upstream_rq_retry_overflow	ClusterName、TaskDefinitionFamily
envoy_server_live	ClusterName、TaskDefinitionFamily
envoy_server_uptime	ClusterName、TaskDefinitionFamily
envoy_http_downstream_rq_xx	ClusterName, TaskDefinitionFamily, envoy_http_conn_manager_prefix, envoy_response_code_class ClusterName, TaskDefinitionFamily, envoy_response_code_class

> **Note**
>
> TaskDefinitionFamily は、メッシュの Kubernetes 名前空間です。

envoy_http_conn_manager_prefix の値は、ingress、egress、または admin のいずれかです。
envoy_response_code_class の値は、 1 (1xx を意味する)、2 (2xx を意味する)、3 (3xx を意味する)、 4 (4xx を意味する)、 5 (5xx を意味する) のいずれかです。

NGINX の Prometheus メトリクス

次のメトリクスは、Amazon EKS および Kubernetes クラスターの NGINX から自動的に収集されます。

メトリクス名	ディメンション	
nginx_ingress_controller_nginx_process_cpu_seconds_total	ClusterName、Namespace 、Service	
nginx_ingress_controller_success	ClusterName、Namespace 、Service	
nginx_ingress_controller_requests	ClusterName、Namespace 、Service	
nginx_ingress_controller_nginx_process_connections	ClusterName、Namespace 、Service	
nginx_ingress_controller_ng	ClusterName、Namespace 、Service	

メトリクス名	ディメンション	
inx_proce ss_connec tions_total		
nginx_ing ress_cont roller_ng inx_proce ss_reside nt_memory _bytes	ClusterName、Namespace 、Service	
nginx_ing ress_cont roller_co nfig_last _reload_s uccessful	ClusterName、Namespace 、Service	
nginx_ing ress_cont roller_re quests	ClusterName、Namespace 、Service、ステータス	

memcached の Prometheus メトリクス

以下のメトリクスは、Amazon EKS および Kubernetes クラスターの memcached から自動的に収集されます。

メトリクス名	ディメンション	
memcached _current_items	ClusterName、Namespace 、Service	

メトリクス名	ディメンション
memcached_current_connections	ClusterName、Namespace 、Service
memcached_limit_bytes	ClusterName、Namespace 、Service
memcached_current_bytes	ClusterName、Namespace 、Service
memcached_written_bytes_total	ClusterName、Namespace 、Service
memcached_read_bytes_total	ClusterName、Namespace 、Service
memcached_items_evicted_total	ClusterName、Namespace 、Service
memcached_items_reclaimed_total	ClusterName、Namespace 、Service
memcached_commands_total	ClusterName、Namespace 、Service ClusterName、Namespace 、Service、コマンド ClusterName、Namespace 、Service、ステータス、コマンド

Java/JMX の Prometheus メトリクス

Amazon EKS および Kubernetes クラスターで収集されたメトリクス

Container Insights は、Amazon EKS および Kubernetes クラスターで JMX Exporter を使用して、Java 仮想マシン (JVM)、Java、Tomcat (Catalina) から、次の定義済みの Prometheus メトリクスを収集できます。詳細については、Github の prometheus/jmx_exporter を参照してください。

Amazon EKS クラスターおよび Kubernetes クラスターでの Java/JMX

メトリクス名	ディメンション	
jvm_classes_loaded	ClusterName , Namespace	
jvm_threads_current	ClusterName , Namespace	
jvm_threads_daemon	ClusterName , Namespace	
java_lang_operatingsystem_totalswapspacesize	ClusterName , Namespace	
java_lang_operatingsystem_systemcpuload	ClusterName , Namespace	
java_lang_operatingsystem_processcpuload	ClusterName , Namespace	
java_lang_operatingsystem_freeswapspacesize	ClusterName , Namespace	

メトリクス名	ディメンション	
java_lang_operatingsystem_totalphysicalmemorysize	ClusterName , Namespace	
java_lang_operatingsystem_freephysicalmemorysize	ClusterName , Namespace	
java_lang_operatingsystem_openfiledescriptorcount	ClusterName , Namespace	
java_lang_operatingsystem_availableprocessors	ClusterName , Namespace	
jvm_memory_bytes_used	ClusterName 、Namespace 、エリア	
jvm_memory_pool_bytes_used	ClusterName 、Namespace 、プール	

> **Note**
>
> area ディメンションの値は、heap または nonheap になります。

pool ディメンションの値は Tenured Gen、Compress Class Space、Survivor Space、Eden Space、Code Cache、または Metaspace のいずれかです。

Amazon EKS および Kubernetes クラスター上の TomCAT/JMX

前の表の Java/JMX メトリクスに加えて、Tomcat ワークロードに対して次のメトリクスも収集されます。

メトリクス名	ディメンション	
catalina_manager_activesessions	ClusterName , Namespace	
catalina_manager_rejectedsessions	ClusterName , Namespace	
catalina_globalrequestprocessor_bytesreceived	ClusterName , Namespace	
catalina_globalrequestprocessor_bytessent	ClusterName , Namespace	
catalina_globalrequestprocessor_requestcount	ClusterName , Namespace	

メトリクス名	ディメンション	
catalina_globalrequestprocessor_errorcount	ClusterName , Namespace	
catalina_globalrequestprocessor_processingtime	ClusterName , Namespace	

Amazon ECS クラスターの Java/JMX

メトリクス名	ディメンション	
jvm_classes_loaded	ClusterName , TaskDefinitionFamily	
jvm_threads_current	ClusterName , TaskDefinitionFamily	
jvm_threads_daemon	ClusterName , TaskDefinitionFamily	
java_lang_operatingsystem_totalswapspacesize	ClusterName , TaskDefinitionFamily	
java_lang_operatingsystem_systemcpuload	ClusterName , TaskDefinitionFamily	

メトリクス名	ディメンション	
java_lang_operatingsystem_processcpuload	ClusterName , TaskDefinitionFamily	
java_lang_operatingsystem_freeswapspacesize	ClusterName , TaskDefinitionFamily	
java_lang_operatingsystem_totalphysicalmemorysize	ClusterName , TaskDefinitionFamily	
java_lang_operatingsystem_freephysicalmemorysize	ClusterName , TaskDefinitionFamily	
java_lang_operatingsystem_openfiledescriptorcount	ClusterName , TaskDefinitionFamily	
java_lang_operatingsystem_availableprocessors	ClusterName , TaskDefinitionFamily	

メトリクス名	ディメンション	
jvm_memory_bytes_used	ClusterName 、TaskDefinitionFamily、エリア	
jvm_memory_pool_bytes_used	ClusterName 、TaskDefinitionFamily、プール	

> **Note**
>
> area ディメンションの値は、heap または nonheap になります。
> pool ディメンションの値は Tenured Gen、Compress Class Space、Survivor Space、Eden Space、Code Cache、または Metaspace のいずれかです。

Amazon ECS クラスターでの Tomcat/JMX

前の表の Java/JMX メトリクスに加えて、Amazon ECS クラスターの Tomcat ワークロードに対して次のメトリクスも収集されます。

メトリクス名	ディメンション	
catalina_manager_activesessions	ClusterName , TaskDefinitionFamily	
catalina_manager_rejectedsessions	ClusterName , TaskDefinitionFamily	
catalina_globalrequestprocessor_bytesreceived	ClusterName , TaskDefinitionFamily	

メトリクス名	ディメンション	
catalina_globalrequestprocessor_bytessent	ClusterName , TaskDefinitionFamily	
catalina_globalrequestprocessor_requestcount	ClusterName , TaskDefinitionFamily	
catalina_globalrequestprocessor_errorcount	ClusterName , TaskDefinitionFamily	
catalina_globalrequestprocessor_processingtime	ClusterName , TaskDefinitionFamily	

HAProxy の Prometheus メトリクス

次のメトリクスは、Amazon EKS および Kubernetes クラスターの HAProxy から自動的に収集されます。

収集されるメトリクスは、使用している HAProxy Ingress のバージョンによって異なります。HAProxy Ingress とそのバージョンの詳細については、「haproxy-ingress」をご参照ください。

メトリクス名	ディメンション	可用性
haproxy_backend_bytes_in_total	ClusterName 、Namespace 、Service	HAProxy Ingress のすべてのバージョン
haproxy_backend_bytes_out_total	ClusterName 、Namespace 、Service	HAProxy Ingress のすべてのバージョン
haproxy_backend_connection_errors_total	ClusterName 、Namespace 、Service	HAProxy Ingress のすべてのバージョン
haproxy_backend_connections_total	ClusterName 、Namespace 、Service	HAProxy Ingress のすべてのバージョン
haproxy_backend_current_sessions	ClusterName 、Namespace 、Service	HAProxy Ingress のすべてのバージョン
haproxy_backend_http_responses_total	ClusterName 、Namespace 、Service、コード、バックエンド	HAProxy Ingress のすべてのバージョン
haproxy_backend_status	ClusterName 、Namespace 、Service	HAProxy Ingress のバージョン 0.10 以降のみ
haproxy_backend_up	ClusterName 、Namespace 、Service	0.10 より前のバージョンの HAProxy Ingress のみ

メトリクス名	ディメンション	可用性
haproxy_frontend_bytes_in_total	ClusterName 、Namespace 、Service	HAProxy Ingress のすべてのバージョン
haproxy_frontend_bytes_out_total	ClusterName 、Namespace 、Service	HAProxy Ingress のすべてのバージョン
haproxy_frontend_connections_total	ClusterName 、Namespace 、Service	HAProxy Ingress のすべてのバージョン
haproxy_frontend_current_sessions	ClusterName 、Namespace 、Service	HAProxy Ingress のすべてのバージョン
haproxy_frontend_http_requests_total	ClusterName 、Namespace 、Service	HAProxy Ingress のすべてのバージョン
haproxy_frontend_http_responses_total	ClusterName 、Namespace 、Service、コード、フロントエンド	HAProxy Ingress のすべてのバージョン
haproxy_frontend_request_errors_total	ClusterName 、Namespace 、Service	HAProxy Ingress のすべてのバージョン

メトリクス名	ディメンション	可用性
haproxy_frontend_requests_denied_total	ClusterName 、Namespace 、Service	HAProxy Ingress のすべてのバージョン

> **Note**
>
> code ディメンションの値は 1xx、2xx、3xx、4xx、5xx、または other のいずれかです。
> backend ディメンションの値は次とすることができます:
>
> - HAProxy Ingress バージョン 0.0.27 以前用の http-default-backend、http-shared-backend、または httpsback-shared-backend。
> - _default_backend0.0.27 より後のバージョンの HAProxy Ingress の 。
>
> frontend ディメンションの値は次とすることができます:
>
> - HAProxy Ingress バージョン 0.0.27 以前用の httpfront-default-backend、httpfront-shared-frontend、または httpfronts。
> - 0.0.27 より後のバージョンの HAProxy Ingress の _front_http または _front_https。

Prometheus メトリクスの表示

App Mesh、NGINX、Java/JMX、Memcached、HAProxy、およびユーザーが追加した手動設定の他の Prometheus エクスポーターから選別された事前集計メトリクスを含む、すべての Prometheus メトリクスについてモニターリングし、アラームを受けることができます。他の Prometheus エクスポーターからのメトリクス収集の詳細については、「新しい Prometheus スクレイピングターゲットを追加するためのチュートリアル: Prometheus API サーバーメトリクス」を参照してください。

CloudWatch コンソールでは、Container Insights は次の事前作成済みのレポートを提供します。

- Amazon EKS および Kubernetes クラスターの場合、App Mesh、NGINX、HAProxy、Memcached、Java/JMX 用の事前作成済みのレポートが提供されます。
- Amazon ECS クラスターの場合、App Mesh と Java/JMX 用の事前作成済みレポートが提供されます。

また、Container Insights は、Container Insights が厳選したメトリクスを収集するワークロードごとにカスタムダッシュボードを提供します。これらのダッシュボードは GitHub からダウンロードすることができます

Prometheus メトリクスをすべて表示するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで [Metrics (メトリクス)] を選択します。
3. 名前空間のリストで、[ContainerInsights/Prometheus] または [ECS/ContainerInsights/Prometheus] を選択します。
4. 次のリストで、ディメンションのセットの 1 つを選択します。次に、表示するメトリクスの横にあるチェックボックスを選択します。

Prometheus メトリクスに関する事前作成済みのレポートを表示するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで、[パフォーマンスのモニターリング] を選択します。
3. ページの上部近くのドロップダウンボックスで、Prometheus オプションのいずれかを選択します。

   もう 1 つのドロップダウンボックスで、表示するクラスターを選択します。

また、NGINX、App Mesh、Memcached、HAProxy、Java/JMX 用のカスタムダッシュボードも用意されています。

Amazon が用意したカスタムダッシュボードを使用するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで、ダッシュボードを選択します。

3. [ダッシュボードの作成] を選択します。新しいダッシュボードの名前を入力し、[ダッシュボードの作成] を選択します。
4. [このダッシュボードに追加] で、[キャンセル] を選択します。
5. [アクション]、[ソースの表示/編集] を選択します。
6. 次の JSON ファイルのいずれかをダウンロードします。

   - Github の NGINX カスタムダッシュボードソース。
   - Github の App Mesh カスタムダッシュボードソース。
   - Github の Memcached カスタムダッシュボードソース。
   - Github の HAProxy-Ingress カスタムダッシュボードソース
   - Github の Java/JMX カスタムダッシュボードソース。
7. ダウンロードした JSON ファイルをテキストエディタで開き、次の変更を加えます。

   - すべての {{YOUR_CLUSTER_NAME}} 文字列をクラスターの正確な名前に置き換えます。テキストの前後に空白を追加しないようにしてください。
   - すべての {{YOUR_REGION}} 文字列を、クラスターが実行されている AWS リージョンに置き換えます。例えば、**us-west-1** テキストの前後に空白を追加しないようにしてください。
   - すべての {{YOUR_NAMESPACE}} 文字列を、ワークロードとまったく同じ名前空間に置き換えます。
   - すべての {{YOUR_SERVICE_NAME}} 文字列を、ワークロードとまったく同じサービス名に置き換えます。例: **haproxy-haproxy-ingress-controller-metrics**
8. JSON BLOB 全体をコピーし、CloudWatch コンソールのテキストボックスに貼り付けて、ボックスに既に入力されている内容を置き換えます。
9. [更新]、[ダッシュボードの保存] の順に選択します。

Prometheus メトリクスのトラブルシューティング

このセクションでは、Prometheus メトリクス設定のトラブルシューティングに役立つ情報を提供します。

トピック

- Amazon ECS での Prometheus メトリクスのトラブルシューティング
- Amazon EKS および Kubernetes クラスターでの Prometheus メトリクスのトラブルシューティング

Amazon ECS での Prometheus メトリクスのトラブルシューティング

このセクションでは、クラスターの Amazon ECS での Prometheus メトリクス設定のトラブルシューティングに役立つ情報を提供します。

CloudWatch Logs に送信された Prometheus メトリクスが表示されません

Prometheus メトリクスを ロググループ /aws/ecs/containerinsights/cluster-name/Prometheus にログイベントとして取り込む必要があります。ロググループが作成されていない場合、または Prometheus メトリクスがロググループに送信されない場合は、Prometheus ターゲットが CloudWatch エージェントによって正常に検出されているかどうかを確認する必要があります。次に、CloudWatch エージェントのセキュリティグループとアクセス許可設定を確認します。次のステップは、デバッグを実行するためのガイドです。

ステップ 1: CloudWatch エージェントのデバッグモードを有効にする

最初に、AWS CloudFormation テンプレートファイル、`cwagent-ecs-prometheus-metric-for-bridge-host.yaml` または `cwagent-ecs-prometheus-metric-for-awsvpc.yaml` に次の太字行を追加して、CloudWatch エージェントをデバッグモードに変更します。その後、ファイルを保存します。

```
cwagentconfig.json: |
    {
      "agent": {
        "debug": true
      },
      "logs": {
        "metrics_collected": {
```

既存のスタックに対して新しい AWS CloudFormation 変更セットを作成します。変更セットの他のパラメータを、既存の AWS CloudFormation スタックと同じ値に設定します。次の例は、EC2 起動タイプとブリッジネットワークモードを使用して Amazon ECS クラスターにインストールされた CloudWatch エージェントです。

```
ECS_NETWORK_MODE=bridge
 CREATE_IAM_ROLES=True
ECS_TASK_ROLE_NAME=your_selected_ecs_task_role_name
ECS_EXECUTION_ROLE_NAME=your_selected_ecs_execution_role_name
NEW_CHANGESET_NAME=your_selected_ecs_execution_role_name
```

```
aws cloudformation create-change-set --stack-name CWAgent-Prometheus-ECS-
${ECS_CLUSTER_NAME}-EC2-${ECS_NETWORK_MODE} \
    --template-body file://cwagent-ecs-prometheus-metric-for-bridge-host.yaml \
    --parameters ParameterKey=ECSClusterName,ParameterValue=$ECS_CLUSTER_NAME \
                 ParameterKey=CreateIAMRoles,ParameterValue=$CREATE_IAM_ROLES \
                 ParameterKey=ECSNetworkMode,ParameterValue=$ECS_NETWORK_MODE \
                 ParameterKey=TaskRoleName,ParameterValue=$ECS_TASK_ROLE_NAME \
                 ParameterKey=ExecutionRoleName,ParameterValue=$ECS_EXECUTION_ROLE_NAME
 \
    --capabilities CAPABILITY_NAMED_IAM \
    --region $AWS_REGION \
    --change-set-name $NEW_CHANGESET_NAME
```

AWS CloudFormation コンソールに移動して、新しい変更セット、$NEW_CHANGESET_NAME を確認します。CWAgentConfigSSMParameter リソースには、1 つの変更が適用されている必要があります。次のコマンドを入力して、変更セットを実行し、CloudWatch エージェントタスクを再起動します。

```
aws ecs update-service --cluster $ECS_CLUSTER_NAME \
--desired-count 0 \
--service your_service_name_here \
--region $AWS_REGION
```

10 秒ほど待ってから、次のコマンドを入力します。

```
aws ecs update-service --cluster $ECS_CLUSTER_NAME \
--desired-count 1 \
--service your_service_name_here \
--region $AWS_REGION
```

ステップ 2: ECS サービス検出ログを確認する

CloudWatch エージェントの ECS タスク定義では、以下のセクションのログがデフォルトで有効になります。ログは、ロググループ /ecs/ecs-cwagent-prometheus の CloudWatch Logs に送信されます。

```
LogConfiguration:
  LogDriver: awslogs
    Options:
      awslogs-create-group: 'True'
      awslogs-group: "/ecs/ecs-cwagent-prometheus"
```

```
      awslogs-region: !Ref AWS::Region
      awslogs-stream-prefix: !Sub 'ecs-${ECSLaunchType}-awsvpc'
```

次の例に示すように、文字列 ECS_SD_Stats でログをフィルタリングして、ECS サービスの検出に関連するメトリクスを取得します。

```
2020-09-1T01:53:14Z D! ECS_SD_Stats: AWSCLI_DescribeContainerInstances: 1
2020-09-1T01:53:14Z D! ECS_SD_Stats: AWSCLI_DescribeInstancesRequest: 1
2020-09-1T01:53:14Z D! ECS_SD_Stats: AWSCLI_DescribeTaskDefinition: 2
2020-09-1T01:53:14Z D! ECS_SD_Stats: AWSCLI_DescribeTasks: 1
2020-09-1T01:53:14Z D! ECS_SD_Stats: AWSCLI_ListTasks: 1
2020-09-1T01:53:14Z D! ECS_SD_Stats: Exporter_DiscoveredTargetCount: 1
2020-09-1T01:53:14Z D! ECS_SD_Stats: LRUCache_Get_EC2MetaData: 1
2020-09-1T01:53:14Z D! ECS_SD_Stats: LRUCache_Get_TaskDefinition: 2
2020-09-1T01:53:14Z D! ECS_SD_Stats: LRUCache_Size_ContainerInstance: 1
2020-09-1T01:53:14Z D! ECS_SD_Stats: LRUCache_Size_TaskDefinition: 2
2020-09-1T01:53:14Z D! ECS_SD_Stats: Latency: 43.399783ms
```

特定の ECS サービス検出サイクルの各メトリクスの意味は次のとおりです。

- AWSCLI_DescribeContainerInstances – 実行された `ECS::DescribeContainerInstances` API 呼び出しの数。
- AWSCLI_DescribeInstancesRequest – 実行された `ECS::DescribeInstancesRequest` API 呼び出しの数。
- AWSCLI_DescribeTaskDefinition – 実行された `ECS::DescribeTaskDefinition` API 呼び出しの数。
- AWSCLI_DescribeTasks – 実行された `ECS::DescribeTasks` API 呼び出しの数。
- AWSCLI_ListTasks – 実行された `ECS::ListTasks` API 呼び出しの数。
- ExporterDiscoveredTargetCount – 検出され、コンテナ内のターゲット結果ファイルに正常にエクスポートされた Prometheus ターゲットの数。
- LRUCache_Get_EC2MetaData – コンテナインスタンスのメタデータがキャッシュから取得された回数。
- LRUCache_Get_TaskDefinition – ECS タスク定義メタデータがキャッシュから取得された回数。
- LRUCache_Size_ContainerInstance – メモリにキャッシュされた一意のコンテナインスタンスのメタデータの数。
- LRUCache_Size_TaskDefinition – メモリにキャッシュされた一意の ECS タスク定義の数。
- レイテンシー – サービス検出サイクルに要する待ち時間。

`ExporterDiscoveredTargetCount` の値をチェックして、検出された Prometheus のターゲットが期待と一致しているかどうかを確認します。そうでない場合、考えられる理由は次のとおりです。

- ECS サービス検出の設定が、アプリケーションの設定と一致していない可能性があります。Docker ラベルベースのサービス検出では、ターゲットコンテナに自動検出するために必要な Docker ラベルが CloudWatch エージェントで構成されていない可能性があります。ECS タスク定義 ARN 正規表現ベースのサービス検出の場合、CloudWatch エージェントの regex 設定がアプリケーションのタスク定義と一致していないことがあります。
- CloudWatch エージェントの ECS タスクロールに、ECS タスクのメタデータを取得するアクセス許可がない可能性があります。CloudWatch エージェントに次の読み取り専用アクセス許可が付与されていることを確認します。
  - `ec2:DescribeInstances`
  - `ecs:ListTasks`
  - `ecs:DescribeContainerInstances`
  - `ecs:DescribeTasks`
  - `ecs:DescribeTaskDefinition`

ステップ 3: ネットワーク接続と ECS タスクのロールポリシーを確認する

`Exporter_DiscoveredTargetCount` の値が Prometheus ターゲットが検出されたことを示しているにもかかわらず、ターゲット CloudWatch Logs ロググループに送信されるログイベントがない場合は、次のいずれかが原因で発生している可能性があります。

- CloudWatch エージェントが Prometheus ターゲットポートに接続できない可能性があります。CloudWatch エージェントの背後にあるセキュリティグループ設定を確認します。プライベート IP は、CloudWatch エージェントが Prometheus エクスポーターポートに接続できるようにする必要があります。
- CloudWatch エージェントの ECS タスクロールに CloudWatchAgentServerPolicy マネージドポリシーがない可能性があります。Prometheus メトリクスをログイベントとして送信できるようにするには、CloudWatch エージェントの ECS タスクロールにこのポリシーが必要です。サンプル AWS CloudFormation テンプレートを使用して IAM ロールを自動的に作成した場合、ECS タスクロールと ECS 実行ロールの両方に、Prometheus モニターリングを実行するための最小権限が付与されます。

Amazon EKS および Kubernetes クラスターでの Prometheus メトリクスのトラブルシューティング

このセクションでは、Amazon EKS および Kubernetes クラスターでの Prometheus メトリクス設定のトラブルシューティングに役立つ情報を提供します。

Amazon EKS の一般的なトラブルシューティングステップ

CloudWatch エージェントが実行中であることを確認するには、次のコマンドを使用します。

```
kubectl get pod -n amazon-cloudwatch
```

出力には、`cwagent-prometheus-id` 列の `NAME` と、`Running` の `STATUS column.` の行を含める必要があります。

実行中のポッドの詳細を表示するには、次のコマンドを入力します。*pod-name* は、`cw-agent-prometheus` で始まる名前を持つポッドの完全名に置き換えます。

```
kubectl describe pod pod-name -n amazon-cloudwatch
```

CloudWatch Container Insights がインストールされている場合は、CloudWatch Logs Insights を使用して、Prometheus メトリクスを収集する CloudWatch エージェントからログをクエリできます。

アプリケーションログをクエリするには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで、[CloudWatch Logs Insights] を選択します。
3. アプリケーションログのロググループ /aws/containerinsights/**cluster-name**/application を選択します。
4. 検索クエリ式を次のクエリに置き換え、[クエリの実行] を選択します。

```
fields ispresent(kubernetes.pod_name) as haskubernetes_pod_name, stream,
 kubernetes.pod_name, log |
filter haskubernetes_pod_name and kubernetes.pod_name like /cwagent-prometheus
```

また、Prometheus メトリクスとメタデータが CloudWatch Logs イベントとして取り込まれていることを確認できます。

Prometheus のデータが取り込まれていることを確認するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで、[CloudWatch Logs Insights] を選択します。
3. [/aws/containerinsights/***cluster-name***/prometheus] を選択します。
4. 検索クエリ式を次のクエリに置き換え、[クエリの実行] を選択します。

```
fields @timestamp, @message | sort @timestamp desc | limit 20
```

削除された Prometheus メトリクスのログ記録

このリリースでは、ヒストグラムタイプの Prometheus メトリクスは収集されません。CloudWatch エージェントを使用して、Prometheus メトリクスがヒストグラムメトリクスであるために削除されているかどうかを確認できます。また、ヒストグラムメトリクスであるために、削除されて CloudWatch に送信されない最初の 500 個の Prometheus メトリクスのリストを記録することもできます。

メトリクスが削除されているかどうかを確認するには、次のコマンドを入力します。

```
kubectl logs -l "app=cwagent-prometheus" -n amazon-cloudwatch --tail=-1
```

メトリクスが削除されている場合は、`/opt/aws/amazon-cloudwatch-agent/logs/amazon-cloudwatch-agent.log` ファイルに次の行が表示されます。

```
I! Drop Prometheus metrics with unsupported types. Only Gauge, Counter and Summary are
 supported.
I! Please enable CWAgent debug mode to view the first 500 dropped metrics
```

これらの行が表示されている場合に、どのメトリクスが削除されているかを確認するには、次のステップを実行します。

削除された Prometheus メトリクスのリストを記録するには

1. `prometheus-eks.yaml` または `prometheus-k8s.yaml` ファイルに次の太字行を追加して、CloudWatch エージェントをデバッグモードに変更し、ファイルを保存します。

```
{
```

```
      "agent": {
        "debug": true
      },
```

ファイルのこのセクションは、次のようになります。

```
cwagentconfig.json: |
    {
      "agent": {
        "debug": true
      },
      "logs": {
        "metrics_collected": {
```

2. 次のコマンドを入力して、CloudWatch エージェントを再インストールしてデバッグモードを有効にします。

```
kubectl delete deployment cwagent-prometheus -n amazon-cloudwatch
kubectl apply -f prometheus.yaml
```

削除されたメトリクスは、CloudWatch エージェントポッドに記録されます。

3. CloudWatch エージェントポッドからログを取得するには、次のコマンドを入力します。

```
kubectl logs -l "app=cwagent-prometheus" -n amazon-cloudwatch --tail=-1
```

または、Container Insights Fluentd ログ記録がインストールされている場合、ログは CloudWatch Logs ロググループ /aws/containerinsights/***cluster_name***/application にも保存されます。

これらのログをクエリするには、「Amazon EKS の一般的なトラブルシューティングステップ」のアプリケーションログをクエリするステップに従います。

Prometheus メトリクスが CloudWatch Logs ログイベントとして取り込まれる場所

CloudWatch エージェントは、Prometheus スクレイプジョブ設定ごとにログストリームを作成します。例えば、`prometheus-eks.yaml` および `prometheus-k8s.yaml` ファイルでは、`job_name: 'kubernetes-pod-appmesh-envoy'` 行が App Mesh メトリクスをスクレイプします。Prometheus ターゲットは `kubernetes-pod-appmesh-envoy` と定義されます。したがって、すべての App Mesh Prometheus メトリクスは、/aws/containerinsights/cluster-name/

Prometheus という名前のロググループの下のログストリーム kubernetes-pod-appmesh-envoy に CloudWatch Logs イベントとして取り込まれます。

CloudWatch メトリクスに Amazon EKS または Kubernetes Prometheus メトリクスが表示されない

まず、Prometheus メトリクスがロググループ /aws/containerinsights/cluster-name/Prometheus にログイベントとして取り込まれることを確認します。「Prometheus メトリクスが CloudWatch Logs ログイベントとして取り込まれる場所」の情報を使用して、ターゲットログストリームを確認します。ログストリームが作成されない場合、またはログストリームに新しいログイベントがない場合は、次の点を確認します。

- Prometheus メトリクスエクスポーターのエンドポイントが正しく設定されていることを確認します。
- CloudWatch エージェント YAML ファイルの `config map: cwagent-prometheus` セクションにある Prometheus スクレイピング設定が正しいことを確認します。この設定は、Prometheus 設定ファイルと同じになります。詳細については、Prometheus ドキュメントの「<scrape_config>」を参照してください。

Prometheus メトリクスがログイベントとして正しく取り込まれる場合は、埋め込みメトリクス形式設定がログイベントに追加され、CloudWatch メトリクスを生成することを確認します。

```
"CloudWatchMetrics":[
   {
      "Metrics":[
         {
            "Name":"envoy_http_downstream_cx_destroy_remote_active_rq"
         }
      ],
      "Dimensions":[
         [
            "ClusterName",
            "Namespace"
         ]
      ],
      "Namespace":"ContainerInsights/Prometheus"
   }
],
```

埋め込みメトリックフォーマットの詳細については、「仕様: 埋め込みメトリクスフォーマット 」を参照してください。

ログイベントにメトリクス形式が埋め込まれていない場合は、CloudWatch エージェントインストール YAML ファイルの `config map: prometheus-cwagentconfig` セクションで、`metric_declaration` セクションが正しく設定されていることを確認してください。(詳細については、新しい Prometheus スクレイピングターゲットを追加するためのチュートリアル: Prometheus API サーバーメトリクス を参照してください)。

Application Insights との統合

Amazon CloudWatch Application Insights は、アプリケーションのモニターリングに役立ち、アプリケーションリソースとテクノロジースタック全体の主要なメトリクス、ログ、アラームを識別して設定します。詳細については、「Amazon CloudWatch Application Insights」を参照してください。

Application Insights を有効にして、コンテナ化されたアプリケーションとマイクロサービスの追加データを収集できます。まだ有効にしていない場合は、Container Insights ダッシュボードのパフォーマンスビューの下にある [Auto-configure Application Insights] (Application Insights の自動設定) を選択して有効にできます。

コンテナ化されたアプリケーションをモニターリングするように CloudWatch Application Insights を既に設定している場合は、Container Insights ダッシュボードの下に Application Insights ダッシュボードが表示されます。

Application Insights とコンテナ化されたアプリケーションの詳細については、「Amazon ECS および Amazon EKS リソースのモニターリングで Application Insights を有効にする」を参照してください。

Container Insights で Amazon ECS ライフサイクルイベントを表示する

Amazon ECS ライフサイクルイベントは、Container Insights のコンソールで表示できます。これにより、コンテナのメトリクス、ログ、およびイベントを単一のビュー内で関連付けることができ、運用上の可視性がより完全なものになります。

イベントには、コンテナインスタンスの状態変更イベント、タスク状態変更イベント、サービスアクションイベントなどがあります。これらは Amazon ECS によって Amazon EventBridge に自動的に送信され、CloudWatch にもイベントログ形式で収集されます。これらのイベントの詳細については、「Amazon ECS イベント」を参照してください。

Amazon ECS ライフサイクルイベントには、標準の Container Insights 料金が適用されます。詳細については、「Amazon CloudWatch 料金表」をご覧ください。

ライフサイクルイベントのテーブルを設定し、クラスターのルールを作成するには、events:PutRule、events:PutTargets、および logs:CreateLogGroup のアクセス許可が必要です。また、EventBridge がログストリームを作成して CloudWatch Logs にログを送信できるリソースベースポリシーがあることを確認する必要があります。リソースポリシーが存在しない場合は、次のコマンドを入力して作成できます。

```
aws --region region logs put-resource-policy --policy-name 'EventBridgeCloudWatchLogs'
 --policy-document '{
    "Statement": [
        {
            "Action": [
                "logs:CreateLogStream",
                "logs:PutLogEvents"
            ],
            "Effect": "Allow",
            "Principal": {
                "Service": ["events.amazonaws.com", "delivery.logs.amazonaws.com"]
            },
            "Resource": "arn:aws:logs:region:account-id:log-group:/aws/events/ecs/
containerinsights/*:*",
            "Sid": "TrustEventBridgeToStoreECSLifecycleLogEvents"
        }
    ],
    "Version": "2012-10-17"
}'
```

次のコマンドを使用して、このポリシーが既に存在しているかどうかを確認し、正しくアタッチされたかどうかを確認できます。

```
aws logs describe-resource-policies --region region --output json
```

ライフサイクルイベントのテーブルを表示するには、events:DescribeRule、events:ListTargetsByRule、および logs:DescribeLogGroups のアクセス許可が必要です。

CloudWatch Container Insights のコンソールで Amazon ECS ライフサイクルイベントを表示するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. [Insights] (インサイト)、[Container Insights] を選択します。

3. [パフォーマンスダッシュボードの表示] を選択します。
4. 次のドロップダウンで、[ECS Clusters] (ECS クラスター)、[ECS Services] (ECS サービス)、または [ECS Tasks] (ECS タスク) のいずれかを選択します。
5. 前のステップで [ECS Services] (ECS サービス) または [ECS Tasks] (ECS タスク) を選択した場合は、[Lifecycle events] (ライフサイクルイベント) タブを選択します。
6. ページの下部に [Configure lifecycle events] (ライフサイクルイベントの設定) が表示されている場合は、これを選択してクラスターの EventBridge ルールを作成します。

   これらのイベントは、Container Insights ペインの下と Application Insights セクションの上に表示されます。これらのイベントに対して追加の分析を実行し、追加のビジュアライゼーションを作成するには、ライフサイクルイベントテーブルで [View in Logs Insights] (Logs Insights で表示) を選択します。

Container Insights のトラブルシューティング

以下のセクションは、Container Insights で問題が発生している場合に役立ちます。

Amazon EKS または Kubernetes でのデプロイに失敗しました

エージェントが Kubernetes クラスターに正しくデプロイされない場合は、以下を試してください。

- 次のコマンドを実行して、ポッドのリストを取得します。

```
kubectl get pods -n amazon-cloudwatch
```

- 次のコマンドを実行して、出力の下部にあるイベントを確認します。

```
kubectl describe pod pod-name -n amazon-cloudwatch
```

- 次のコマンドを実行して、ログを確認します。

```
kubectl logs pod-name -n amazon-cloudwatch
```

無許可パニック: kubelet から cadvisor データを取得できない

デプロイがエラー `Unauthorized panic: Cannot retrieve cadvisor data from kubelet` で失敗する場合は、kubelet で Webhook 認証モードが有効になっていない可能性がありま

す。このモードでは、Container Insights に必要です。詳細については、「前提条件を確認する」を参照してください。

Amazon ECS 上の削除および再作成されたクラスターへの Container Insights のデプロイ

Container Insights が有効になっていない既存の Amazon ECS クラスターを削除し、同じ名前で再作成した場合、再作成時にこの新しいクラスターで Container Insights を有効にすることはできません。再作成して有効にするには、次のコマンドを入力します。

```
aws ecs update-cluster-settings --cluster myCICluster --settings
 name=containerInsights,value=enabled
```

無効なエンドポイントエラー

次のようなエラーメッセージが表示された場合は、使用しているコマンドの *cluster-name* や *region-name* などのすべてのプレースホルダを、デプロイ用の正しい情報に置き換えていることを確認します。

```
"log": "2020-04-02T08:36:16Z E! cloudwatchlogs: code: InvalidEndpointURL, message:
 invalid endpoint uri, original error: &url.Error{Op:\"parse\", URL:\"https://
logs.{{region_name}}.amazonaws.com/\", Err:\"{\"}, &awserr.baseError{code:
\"InvalidEndpointURL\", message:\"invalid endpoint uri\", errs:[]error{(*url.Error)
(0xc0008723c0)}}\n",
```

メトリクスがコンソールに表示されない

AWS Management Console に Container Insights メトリクスが表示されない場合は、Container Insights のセットアップが完了していることを確認します。メトリクスは、Container Insights が完全にセットアップされるまで表示されません。詳細については、「Container Insights の設定」を参照してください。

クラスターのアップグレード後に Amazon EKS または Kubernetes でポッドメトリクスが欠けている

このセクションは、CloudWatch エージェントをデーモンセットとして新規クラスターまたはアップグレードされたクラスターにデプロイした後に、すべてまたは一部のポッドメトリクスが欠けている場合、またはメッセージ `W! No pod metric collected` のエラーログが表示される場合に役に立ちます。

これらのエラーは、containerd や docker systemd cgroup ドライバーなどのコンテナランタイムの変更によって引き起こされる可能性があります。通常は、デプロイマニフェストを更新して、ホストからの containerd ソケットがコンテナにマウントされるようにすることで、解決できます。次の例を参照してください。

```
# For full example see https://github.com/aws-samples/amazon-cloudwatch-container-
insights/blob/latest/k8s-deployment-manifest-templates/deployment-mode/daemonset/
container-insights-monitoring/cwagent/cwagent-daemonset.yaml
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: cloudwatch-agent
  namespace: amazon-cloudwatch
spec:
  template:
    spec:
      containers:
        - name: cloudwatch-agent
# ...
          # Don't change the mountPath
          volumeMounts:
# ...
            - name: dockersock
              mountPath: /var/run/docker.sock
              readOnly: true
            - name: varlibdocker
              mountPath: /var/lib/docker
              readOnly: true
            - name: containerdsock # NEW mount
              mountPath: /run/containerd/containerd.sock
              readOnly: true
# ...
      volumes:
# ...
        - name: dockersock
          hostPath:
            path: /var/run/docker.sock
        - name: varlibdocker
          hostPath:
            path: /var/lib/docker
        - name: containerdsock # NEW volume
          hostPath:
```

```
            path: /run/containerd/containerd.sock
```

Amazon EKS で Bottlerocket を使用する場合、ポッドメトリクスがない

Bottlerocket は Linux ベースのオープンソースのオペレーティングシステムで、コンテナを実行するために AWS によって専用に構築されています。

Bottlerocket はホスト上で異なる `containerd` パスを使用するため、ボリュームをその場所に変更する必要があります。変更しないと、`W! No pod metric collected` を含むログにエラーが表示されます。次の例を参照してください。

```
volumes:
  # ...
    - name: containerdsock
      hostPath:
        # path: /run/containerd/containerd.sock
        # bottlerocket does not mount containerd sock at normal place
        # https://github.com/bottlerocket-os/bottlerocket/
commit/91810c85b83ff4c3660b496e243ef8b55df0973b
        path: /run/dockershim.sock
```

Amazon EKS または Kubernetes で containerd ランタイムを使用する場合、コンテナファイルシステムのメトリクスがない

これは既知の問題であり、コミュニティの貢献者が取り組んでいます。詳細については、containerd のディスク使用量メトリクスとコンテナファイルシステムのメトリクスは、GitHub の containerd 用 cadvisor ではサポートされていませんを参照してください。

Prometheus メトリクスを収集するときに CloudWatch エージェントから予期しないログボリュームが増える

これは、CloudWatch エージェントのバージョン 1.247347.6b250880 で導入された回帰です。この回帰は、より新しいバージョンのエージェントで既に修正されています。この影響は、お客様が CloudWatch エージェント自体のログを収集し、Prometheus を使用しているシナリオに限定されていました。詳細については、[prometheus] エージェントが GitHub のログにスクレイプされたメトリクスをすべて出力しているを参照してください。

リリースノートに記載されている最新の Docker イメージが Dockerhub から見つからない

実際のリリースを内部的に開始する前に、Github でリリースノートとタグを更新します。Github でバージョン番号を上げてから、レジストリで最新の Docker イメージが表示されるまで、通常 1 ~ 2 週間かかります。CloudWatch エージェントコンテナイメージは夜間にリリースされません。次の場所 (https://github.com/aws/amazon-cloudwatch-agent/tree/main/amazon-cloudwatch-container-insights/cloudwatch-agent-dockerfile) にあるソースから直接イメージを作成できます。

CloudWatch エージェントの CrashLoopBackoff エラー

CloudWatch エージェントの `CrashLoopBackOff` のエラーが表示された場合は、IAM アクセス許可が正しく設定されていることを確認してください。詳細については、「前提条件を確認する」を参照してください。

CloudWatch エージェントまたは Fluentd ポッドが保留状態でスタックする

CloudWatch エージェントまたは Fluentd ポッドが `Pending` 状態でスタックしている場合、または、`FailedScheduling` エラーでスタックしている場合は、エージェントに必要なコア数と RAM の量に基づいて、ノードに十分なコンピューティングリソースがあるかどうかを確認します。以下のコマンドを使用して、ポッドを記述します。

```
kubectl describe pod cloudwatch-agent-85ppg -n amazon-cloudwatch
```

独自の CloudWatch エージェント Docker イメージの構築

https://github.com/aws-samples/amazon-cloudwatch-container-insights/blob/latest/cloudwatch-agent-dockerfile/Dockerfile にある Dockerfile を参照することで、独自の CloudWatch エージェント Docker イメージを構築できます。

Dockerfile では、`docker buildx` を使用して直接マルチアーキテクチャイメージの作成をサポートしています。

コンテナへの他の CloudWatch エージェント機能のデプロイ

CloudWatch エージェントを使用して、コンテナに追加のモニターリング機能をデプロイできます。主な機能は以下のとおりです。

- 埋め込みメトリクス形式 – 詳細については、「ログ内へのメトリクスの埋め込み」を参照してください。

- StatsD – 詳細については、「StatsD を使用してカスタムメトリクスを取得する 」を参照してください。

手順と必要なファイルは、GitHub の次の場所にあります。

- Amazon ECS コンテナについては、「Example Amazon ECS task definitions based on deployment modes」を参照してください。
- Amazon EKS および Kubernetes コンテナについては、「Example Kubernetes YAML files based on deployment modes」を参照してください。

Lambda Insights

CloudWatch Lambda Insights は、 で実行されているサーバーレスアプリケーション用のモニターリングおよびトラブルシューティングソリューションですAWS Lambda このソリューションでは、CPU 時間、メモリ、ディスク、ネットワークなどのシステムレベルのメトリクスが収集、集約、要約されます。また、コールドスタートや Lambda ワーカーシャットダウンなどの診断情報が収集、集約、要約されるため、Lambda 関数に関する問題を特定し、迅速に解決できます。

Lambda Insights は、Lambda レイヤーとして提供される新しい CloudWatch Lambda 拡張機能を使用します。この拡張機能を Lambda 関数にインストールすると、システムレベルのメトリクスを収集し、その Lambda 関数の呼び出しごとに 1 つのパフォーマンスログイベントが発生します。CloudWatch は、埋め込みメトリクスフォーマットを使用して、ログイベントからメトリクスを抽出します。

Lambda 拡張機能の詳細については、「AWS Lambda 拡張機能を使用する」を参照してください。埋め込みメトリックフォーマットの詳細については、「ログ内へのメトリクスの埋め込み」を参照してください。

Lambda Insights は、Lambda 拡張機能をサポートする Lambda ランタイムを使用する任意の Lambda 関数で使用できます。これらのランタイムの一覧については、「Lambda 拡張機能 API」を参照してください。

料金表

Lambda Insights に対して有効になっている各 Lambda 関数の料金は、メトリクスとログに使用した分に対してのみ発生します。料金の例については、「Amazon CloudWatch の料金」を参照してください。

料金は、Lambda 拡張機能の実行時間 (1 ミリ秒単位) に対して課金されます。Lambda の料金の詳細については、「AWS Lambda の料金」を参照してください。

Lambda Insights の使用を開始する

Lambda 関数で Lambda Insights を有効にするには、Lambda コンソールでワンクリックトグルを使用できます。または、AWS CLI、AWS CloudFormation、AWS Serverless Application Model CLI、または AWS Cloud Development Kit (AWS CDK)を使用できます。

以下のセクションでは、これらのステップを完了するための詳細な手順について説明します。

トピック

Lambda Insights 拡張機能の利用可能なバージョン

このセクションでは、Lambda Insights 拡張機能のバージョンと、各 AWS リージョンでこれらの拡張機能に使用する ARN の一覧を示します。

トピック

x86-64 プラットフォーム

このセクションでは、x86-64 プラットフォーム向け Lambda Insights 拡張機能のバージョンと、これらの拡張機能が各 AWS リージョンで使用する ARN の一覧を示します。

> **Important**
>
> Lambda Insights の拡張機能、1.0.317.0 以降は、Amazon Linux 1 をサポートしていません。

1.0.317.0

バージョン 1.0.317.0 には、Amazon Linux 1 プラットフォームのサポートの削除とバグ修正が含まれています。AWS GovCloud (US) リージョンのサポートも含まれています。

バージョン 1.0.317.0 の ARN

次の表に、利用可能な各 AWS リージョンでこのバージョンの拡張機能で使用する ARN の一覧を示します。

リージョン	ARN
米国東部 (バージニア北部)	arn:aws:lambda:us-east-1:580247275435:layer:LambdaInsightsExtension:52
米国東部 (オハイオ)	arn:aws:lambda:us-east-2:580247275435:layer:LambdaInsightsExtension:52
米国西部 (北カリフォルニア)	arn:aws:lambda:us-west-1:580247275435:layer:LambdaInsightsExtension:52
米国西部 (オレゴン)	arn:aws:lambda:us-west-2:580247275435:layer:LambdaInsightsExtension:52
アフリカ (ケープタウン)	arn:aws:lambda:af-south-1:012438385374:layer:LambdaInsightsExtension:43
アジアパシフィック (香港)	arn:aws:lambda:ap-east-1:519774774795:layer:LambdaInsightsExtension:43
アジアパシフィック (ハイデラバード)	arn:aws:lambda:ap-south-2:891564319516:layer:LambdaInsightsExtension:25
アジアパシフィック (ジャカルタ)	arn:aws:lambda:ap-southeast-3:439286490199:layer:LambdaInsightsExtension:29

リージョン	ARN
アジアパシフィック (メルボルン)	arn:aws:lambda:ap-southeast-4:158895979263:layer:LambdaInsightsExtension:20
アジアパシフィック (ムンバイ)	arn:aws:lambda:ap-south-1:580247275435:layer:LambdaInsightsExtension:50
アジアパシフィック (大阪)	arn:aws:lambda:ap-northeast-3:194566237122:layer:LambdaInsightsExtension:33
アジアパシフィック (ソウル)	arn:aws:lambda:ap-northeast-2:580247275435:layer:LambdaInsightsExtension:51
アジアパシフィック (シンガポール)	arn:aws:lambda:ap-southeast-1:580247275435:layer:LambdaInsightsExtension:52
アジアパシフィック (シドニー)	arn:aws:lambda:ap-southeast-2:580247275435:layer:LambdaInsightsExtension:52
アジアパシフィック (東京)	arn:aws:lambda:ap-northeast-1:580247275435:layer:LambdaInsightsExtension:79
カナダ (中部)	arn:aws:lambda:ca-central-1:580247275435:layer:LambdaInsightsExtension:51
カナダ西部 (カルガリー)	arn:aws:lambda:ca-west-1:946466191631:layer:LambdaInsightsExtension:12
中国 (北京)	arn:aws-cn:lambda:cn-north-1:488211338238:layer:LambdaInsightsExtension:42
中国 (寧夏)	arn:aws-cn:lambda:cn-northwest-1:488211338238:layer:LambdaInsightsExtension:42
欧州 (フランクフルト)	arn:aws:lambda:eu-central-1:580247275435:layer:LambdaInsightsExtension:52
欧州 (アイルランド)	arn:aws:lambda:eu-west-1:580247275435:layer:LambdaInsightsExtension:52

リージョン	ARN
欧州 (ロンドン)	arn:aws:lambda:eu-west-2:580247275435:layer:LambdaInsightsExtension:52
欧州 (ミラノ)	arn:aws:lambda:eu-south-1:339249233099:layer:LambdaInsightsExtension:43
欧州 (パリ)	arn:aws:lambda:eu-west-3:580247275435:layer:LambdaInsightsExtension:51
欧州 (スペイン)	arn:aws:lambda:eu-south-2:352183217350:layer:LambdaInsightsExtension:27
欧州 (ストックホルム)	arn:aws:lambda:eu-north-1:580247275435:layer:LambdaInsightsExtension:49
欧州 (チューリッヒ)	arn:aws:lambda:eu-central-2:033019950311:layer:LambdaInsightsExtension:26
イスラエル (テルアビブ)	arn:aws:lambda:il-central-1:459530977127:layer:LambdaInsightsExtension:20
中東 (バーレーン)	arn:aws:lambda:me-south-1:285320876703:layer:LambdaInsightsExtension:43
中東 (アラブ首長国連邦)	arn:aws:lambda:me-central-1:732604637566:layer:LambdaInsightsExtension:26
南米 (サンパウロ)	arn:aws:lambda:sa-east-1:580247275435:layer:LambdaInsightsExtension:51
AWS GovCloud (米国東部)	arn:aws-us-gov:lambda:us-gov-east-1:122132214140:layer:LambdaInsightsExtension:19
AWS GovCloud (米国西部)	arn:aws-us-gov:lambda:us-gov-west-1:751350123760:layer:LambdaInsightsExtension:19

1.0.295.0

バージョン 1.0.295.0 には、互換性のあるすべてのランタイム用の依存関係の更新が含まれています。

バージョン 1.0.295.0 の ARN

次の表に、利用可能な各 AWS リージョンでこのバージョンの拡張機能で使用する ARN の一覧を示します。

リージョン	ARN
米国東部 (バージニア北部)	arn:aws:lambda:us-east-1:580247275435:layer:LambdaInsightsExtension:51
米国東部 (オハイオ)	arn:aws:lambda:us-east-2:580247275435:layer:LambdaInsightsExtension:51
米国西部 (北カリフォルニア)	arn:aws:lambda:us-west-1:580247275435:layer:LambdaInsightsExtension:51
米国西部 (オレゴン)	arn:aws:lambda:us-west-2:580247275435:layer:LambdaInsightsExtension:51
アフリカ (ケープタウン)	arn:aws:lambda:af-south-1:012438385374:layer:LambdaInsightsExtension:42
アジアパシフィック (香港)	arn:aws:lambda:ap-east-1:519774774795:layer:LambdaInsightsExtension:42
アジアパシフィック (ハイデラバード)	arn:aws:lambda:ap-south-2:891564319516:layer:LambdaInsightsExtension:24
アジアパシフィック (ジャカルタ)	arn:aws:lambda:ap-southeast-3:439286490199:layer:LambdaInsightsExtension:28
アジアパシフィック (メルボルン)	arn:aws:lambda:ap-southeast-4:158895979263:layer:LambdaInsightsExtension:19

リージョン	ARN
アジアパシフィック（ムンバイ）	arn:aws:lambda:ap-south-1:580247275435:layer:LambdaInsightsExtension:49
アジアパシフィック (大阪)	arn:aws:lambda:ap-northeast-3:194566237122:layer:LambdaInsightsExtension:32
アジアパシフィック (ソウル)	arn:aws:lambda:ap-northeast-2:580247275435:layer:LambdaInsightsExtension:50
アジアパシフィック (シンガポール)	arn:aws:lambda:ap-southeast-1:580247275435:layer:LambdaInsightsExtension:51
アジアパシフィック (シドニー)	arn:aws:lambda:ap-southeast-2:580247275435:layer:LambdaInsightsExtension:51
アジアパシフィック (東京)	arn:aws:lambda:ap-northeast-1:580247275435:layer:LambdaInsightsExtension:78
カナダ (中部)	arn:aws:lambda:ca-central-1:580247275435:layer:LambdaInsightsExtension:50
カナダ西部 (カルガリー)	arn:aws:lambda:ca-west-1:946466191631:layer:LambdaInsightsExtension:11
中国 (北京)	arn:aws-cn:lambda:cn-north-1:488211338238:layer:LambdaInsightsExtension:41
中国 (寧夏)	arn:aws-cn:lambda:cn-northwest-1:488211338238:layer:LambdaInsightsExtension:41
欧州 (フランクフルト)	arn:aws:lambda:eu-central-1:580247275435:layer:LambdaInsightsExtension:51
欧州 (アイルランド)	arn:aws:lambda:eu-west-1:580247275435:layer:LambdaInsightsExtension:51
欧州 (ロンドン)	arn:aws:lambda:eu-west-2:580247275435:layer:LambdaInsightsExtension:51

リージョン	ARN
欧州 (ミラノ)	arn:aws:lambda:eu-south-1:339249233099:layer:LambdaInsightsExtension:42
欧州 (パリ)	arn:aws:lambda:eu-west-3:580247275435:layer:LambdaInsightsExtension:50
欧州 (スペイン)	arn:aws:lambda:eu-south-2:352183217350:layer:LambdaInsightsExtension:26
欧州 (ストックホルム)	arn:aws:lambda:eu-north-1:580247275435:layer:LambdaInsightsExtension:48
欧州 (チューリッヒ)	arn:aws:lambda:eu-central-2:033019950311:layer:LambdaInsightsExtension:25
イスラエル (テルアビブ)	arn:aws:lambda:il-central-1:459530977127:layer:LambdaInsightsExtension:19
中東 (バーレーン)	arn:aws:lambda:me-south-1:285320876703:layer:LambdaInsightsExtension:42
中東 (アラブ首長国連邦)	arn:aws:lambda:me-central-1:732604637566:layer:LambdaInsightsExtension:25
南米 (サンパウロ)	arn:aws:lambda:sa-east-1:580247275435:layer:LambdaInsightsExtension:50

1.0.275.0

バージョン 1.0.275.0 には、互換性のあるすべてのランタイム用の重要な依存関係の更新が含まれています。

バージョン 1.0.275.0 の ARN

次の表に、利用可能な各 AWS リージョンでこのバージョンの拡張機能で使用する ARN の一覧を示します。

リージョン	ARN
米国東部 (バージニア北部)	arn:aws:lambda:us-east-1:580247275435:layer:LambdaInsightsExtension:49
米国東部 (オハイオ)	arn:aws:lambda:us-east-2:580247275435:layer:LambdaInsightsExtension:49
米国西部 (北カリフォルニア)	arn:aws:lambda:us-west-1:580247275435:layer:LambdaInsightsExtension:49
米国西部 (オレゴン)	arn:aws:lambda:us-west-2:580247275435:layer:LambdaInsightsExtension:49
アフリカ (ケープタウン)	arn:aws:lambda:af-south-1:012438385374:layer:LambdaInsightsExtension:40
アジアパシフィック (香港)	arn:aws:lambda:ap-east-1:519774774795:layer:LambdaInsightsExtension:40
アジアパシフィック (ハイデラバード)	arn:aws:lambda:ap-south-2:891564319516:layer:LambdaInsightsExtension:22
アジアパシフィック (ジャカルタ)	arn:aws:lambda:ap-southeast-3:439286490199:layer:LambdaInsightsExtension:26
アジアパシフィック (メルボルン)	arn:aws:lambda:ap-southeast-4:158895979263:layer:LambdaInsightsExtension:17
アジアパシフィック (ムンバイ)	arn:aws:lambda:ap-south-1:580247275435:layer:LambdaInsightsExtension:47
アジアパシフィック (大阪)	arn:aws:lambda:ap-northeast-3:194566237122:layer:LambdaInsightsExtension:30
アジアパシフィック (ソウル)	arn:aws:lambda:ap-northeast-2:580247275435:layer:LambdaInsightsExtension:48

リージョン	ARN
アジアパシフィック (シンガポール)	arn:aws:lambda:ap-southeast-1:580247275435:layer:LambdaInsightsExtension:49
アジアパシフィック (シドニー)	arn:aws:lambda:ap-southeast-2:580247275435:layer:LambdaInsightsExtension:49
アジアパシフィック (東京)	arn:aws:lambda:ap-northeast-1:580247275435:layer:LambdaInsightsExtension:76
カナダ (中部)	arn:aws:lambda:ca-central-1:580247275435:layer:LambdaInsightsExtension:48
カナダ西部 (カルガリー)	arn:aws:lambda:ca-west-1:946466191631:layer:LambdaInsightsExtension:9
中国 (北京)	arn:aws-cn:lambda:cn-north-1:488211338238:layer:LambdaInsightsExtension:39
中国 (寧夏)	arn:aws-cn:lambda:cn-northwest-1:488211338238:layer:LambdaInsightsExtension:39
欧州 (フランクフルト)	arn:aws:lambda:eu-central-1:580247275435:layer:LambdaInsightsExtension:49
欧州 (アイルランド)	arn:aws:lambda:eu-west-1:580247275435:layer:LambdaInsightsExtension:49
欧州 (ロンドン)	arn:aws:lambda:eu-west-2:580247275435:layer:LambdaInsightsExtension:49
欧州 (ミラノ)	arn:aws:lambda:eu-south-1:339249233099:layer:LambdaInsightsExtension:40
欧州 (パリ)	arn:aws:lambda:eu-west-3:580247275435:layer:LambdaInsightsExtension:48
欧州 (スペイン)	arn:aws:lambda:eu-south-2:352183217350:layer:LambdaInsightsExtension:24

リージョン	ARN
欧州 (ストックホルム)	arn:aws:lambda:eu-north-1:580247275435:layer:LambdaInsightsExtension:46
欧州 (チューリッヒ)	arn:aws:lambda:eu-central-2:033019950311:layer:LambdaInsightsExtension:23
イスラエル (テルアビブ)	arn:aws:lambda:il-central-1:459530977127:layer:LambdaInsightsExtension:17
中東 (バーレーン)	arn:aws:lambda:me-south-1:285320876703:layer:LambdaInsightsExtension:40
中東 (アラブ首長国連邦)	arn:aws:lambda:me-central-1:732604637566:layer:LambdaInsightsExtension:23
南米 (サンパウロ)	arn:aws:lambda:sa-east-1:580247275435:layer:LambdaInsightsExtension:48

1.0.273.0

バージョン 1.0.273.0 では、互換性のあるすべてのランタイムに対して重要なバグ修正を行っているほか、カナダ西部 (カルガリー) リージョンのサポートを追加しています。

バージョン 1.0.273.0 の ARN

次の表に、利用可能な各 AWS リージョンでこのバージョンの拡張機能で使用する ARN の一覧を示します。

リージョン	ARN
米国東部 (バージニア北部)	arn:aws:lambda:us-east-1:580247275435:layer:LambdaInsightsExtension:45
米国東部 (オハイオ)	arn:aws:lambda:us-east-2:580247275435:layer:LambdaInsightsExtension:45

リージョン	ARN
米国西部 (北カリフォルニア)	arn:aws:lambda:us-west-1:580247275435:layer:LambdaInsightsExtension:45
米国西部 (オレゴン)	arn:aws:lambda:us-west-2:580247275435:layer:LambdaInsightsExtension:45
アフリカ (ケープタウン)	arn:aws:lambda:af-south-1:012438385374:layer:LambdaInsightsExtension:35
アジアパシフィック (香港)	arn:aws:lambda:ap-east-1:519774774795:layer:LambdaInsightsExtension:35
アジアパシフィック (ハイデラバード)	arn:aws:lambda:ap-south-2:891564319516:layer:LambdaInsightsExtension:17
アジアパシフィック (ジャカルタ)	arn:aws:lambda:ap-southeast-3:439286490199:layer:LambdaInsightsExtension:21
アジアパシフィック (メルボルン)	arn:aws:lambda:ap-southeast-4:158895979263:layer:LambdaInsightsExtension:12
アジアパシフィック (ムンバイ)	arn:aws:lambda:ap-south-1:580247275435:layer:LambdaInsightsExtension:43
アジアパシフィック (大阪)	arn:aws:lambda:ap-northeast-3:194566237122:layer:LambdaInsightsExtension:26
アジアパシフィック (ソウル)	arn:aws:lambda:ap-northeast-2:580247275435:layer:LambdaInsightsExtension:44
アジアパシフィック (シンガポール)	arn:aws:lambda:ap-southeast-1:580247275435:layer:LambdaInsightsExtension:45
アジアパシフィック (シドニー)	arn:aws:lambda:ap-southeast-2:580247275435:layer:LambdaInsightsExtension:45
アジアパシフィック (東京)	arn:aws:lambda:ap-northeast-1:580247275435:layer:LambdaInsightsExtension:72

リージョン	ARN
カナダ (中部)	arn:aws:lambda:ca-central-1:580247275435:layer:LambdaInsightsExtension:44
カナダ西部 (カルガリー)	arn:aws:lambda:ca-west-1:946466191631:layer:LambdaInsightsExtension:4
中国 (北京)	arn:aws-cn:lambda:cn-north-1:488211338238:layer:LambdaInsightsExtension:36
中国 (寧夏)	arn:aws-cn:lambda:cn-northwest-1:488211338238:layer:LambdaInsightsExtension:36
欧州 (フランクフルト)	arn:aws:lambda:eu-central-1:580247275435:layer:LambdaInsightsExtension:45
欧州 (アイルランド)	arn:aws:lambda:eu-west-1:580247275435:layer:LambdaInsightsExtension:45
欧州 (ロンドン)	arn:aws:lambda:eu-west-2:580247275435:layer:LambdaInsightsExtension:45
欧州 (ミラノ)	arn:aws:lambda:eu-south-1:339249233099:layer:LambdaInsightsExtension:35
欧州 (パリ)	arn:aws:lambda:eu-west-3:580247275435:layer:LambdaInsightsExtension:44
欧州 (スペイン)	arn:aws:lambda:eu-south-2:352183217350:layer:LambdaInsightsExtension:19
欧州 (ストックホルム)	arn:aws:lambda:eu-north-1:580247275435:layer:LambdaInsightsExtension:42
欧州 (チューリッヒ)	arn:aws:lambda:eu-central-2:033019950311:layer:LambdaInsightsExtension:17
イスラエル (テルアビブ)	arn:aws:lambda:il-central-1:459530977127:layer:LambdaInsightsExtension:12

リージョン	ARN
中東 (バーレーン)	arn:aws:lambda:me-south-1:285320876703:layer:LambdaInsightsExtension:35
中東 (アラブ首長国連邦)	arn:aws:lambda:me-central-1:732604637566:layer:LambdaInsightsExtension:18
南米 (サンパウロ)	arn:aws:lambda:sa-east-1:580247275435:layer:LambdaInsightsExtension:44

1.0.229.0

バージョン 1.0.229.0 では、互換性のあるすべてのランタイムに対して重要なバグ修正を行っているほか、イスラエル (テルアビブ) リージョンのサポートを追加しています。

バージョン 1.0.229.0 の ARN

次の表に、利用可能な各 AWS リージョンでこのバージョンの拡張機能で使用する ARN の一覧を示します。

リージョン	ARN
米国東部 (バージニア北部)	arn:aws:lambda:us-east-1:580247275435:layer:LambdaInsightsExtension:38
米国東部 (オハイオ)	arn:aws:lambda:us-east-2:580247275435:layer:LambdaInsightsExtension:38
米国西部 (北カリフォルニア)	arn:aws:lambda:us-west-1:580247275435:layer:LambdaInsightsExtension:38
米国西部 (オレゴン)	arn:aws:lambda:us-west-2:580247275435:layer:LambdaInsightsExtension:38
アフリカ (ケープタウン)	arn:aws:lambda:af-south-1:012438385374:layer:LambdaInsightsExtension:28

リージョン	ARN
アジアパシフィック (香港)	arn:aws:lambda:ap-east-1:519774774795:layer:LambdaInsightsExtension:28
アジアパシフィック (ハイデラバード)	arn:aws:lambda:ap-south-2:891564319516:layer:LambdaInsightsExtension:10
アジアパシフィック (ジャカルタ)	arn:aws:lambda:ap-southeast-3:439286490199:layer:LambdaInsightsExtension:14
アジアパシフィック (メルボルン)	arn:aws:lambda:ap-southeast-4:158895979263:layer:LambdaInsightsExtension:5
アジアパシフィック (ムンバイ)	arn:aws:lambda:ap-south-1:580247275435:layer:LambdaInsightsExtension:36
アジアパシフィック (大阪)	arn:aws:lambda:ap-northeast-3:194566237122:layer:LambdaInsightsExtension:19
アジアパシフィック (ソウル)	arn:aws:lambda:ap-northeast-2:580247275435:layer:LambdaInsightsExtension:37
アジアパシフィック (シンガポール)	arn:aws:lambda:ap-southeast-1:580247275435:layer:LambdaInsightsExtension:38
アジアパシフィック (シドニー)	arn:aws:lambda:ap-southeast-2:580247275435:layer:LambdaInsightsExtension:38
アジアパシフィック (東京)	arn:aws:lambda:ap-northeast-1:580247275435:layer:LambdaInsightsExtension:60
カナダ (中部)	arn:aws:lambda:ca-central-1:580247275435:layer:LambdaInsightsExtension:37
中国 (北京)	arn:aws-cn:lambda:cn-north-1:488211338238:layer:LambdaInsightsExtension:29
中国 (寧夏)	arn:aws-cn:lambda:cn-northwest-1:488211338238:layer:LambdaInsightsExtension:29

リージョン	ARN
欧州 (フランクフルト)	arn:aws:lambda:eu-central-1:580247275435:layer:LambdaInsightsExtension:38
欧州 (アイルランド)	arn:aws:lambda:eu-west-1:580247275435:layer:LambdaInsightsExtension:38
欧州 (ロンドン)	arn:aws:lambda:eu-west-2:580247275435:layer:LambdaInsightsExtension:38
欧州 (ミラノ)	arn:aws:lambda:eu-south-1:339249233099:layer:LambdaInsightsExtension:28
欧州 (パリ)	arn:aws:lambda:eu-west-3:580247275435:layer:LambdaInsightsExtension:37
欧州 (スペイン)	arn:aws:lambda:eu-south-2:352183217350:layer:LambdaInsightsExtension:12
欧州 (ストックホルム)	arn:aws:lambda:eu-north-1:580247275435:layer:LambdaInsightsExtension:35
欧州 (チューリッヒ)	arn:aws:lambda:eu-central-2:033019950311:layer:LambdaInsightsExtension:11
イスラエル (テルアビブ)	arn:aws:lambda:il-central-1:459530977127:layer:LambdaInsightsExtension:5
中東 (バーレーン)	arn:aws:lambda:me-south-1:285320876703:layer:LambdaInsightsExtension:28
中東 (アラブ首長国連邦)	arn:aws:lambda:me-central-1:732604637566:layer:LambdaInsightsExtension:11
南米 (サンパウロ)	arn:aws:lambda:sa-east-1:580247275435:layer:LambdaInsightsExtension:37

1.0.178.0

バージョン 1.0.178.0 では、以下の AWS リージョンのサポートが追加されています。

- アジアパシフィック (ハイデラバード)
- アジアパシフィック (ジャカルタ)
- 欧州 (スペイン)
- 欧州 (チューリッヒ)
- 中東 (アラブ首長国連邦)

バージョン 1.0.178.0 の ARN

次の表に、利用可能な各 AWS リージョンでこのバージョンの拡張機能で使用する ARN の一覧を示します。

リージョン	ARN
米国東部 (バージニア北部)	arn:aws:lambda:us-east-1:580247275435:layer:LambdaInsightsExtension:35
米国東部 (オハイオ)	arn:aws:lambda:us-east-2:580247275435:layer:LambdaInsightsExtension:33
米国西部 (北カリフォルニア)	arn:aws:lambda:us-west-1:580247275435:layer:LambdaInsightsExtension:33
米国西部 (オレゴン)	arn:aws:lambda:us-west-2:580247275435:layer:LambdaInsightsExtension:33
アフリカ (ケープタウン)	arn:aws:lambda:af-south-1:012438385374:layer:LambdaInsightsExtension:25
アジアパシフィック (香港)	arn:aws:lambda:ap-east-1:519774774795:layer:LambdaInsightsExtension:25
アジアパシフィック (ハイデラバード)	arn:aws:lambda:ap-south-2:891564319516:layer:LambdaInsightsExtension:8

リージョン	ARN
アジアパシフィック (ジャカルタ)	arn:aws:lambda:ap-southeast-3:439286490199:layer:LambdaInsightsExtension:11
アジアパシフィック (ムンバイ)	arn:aws:lambda:ap-south-1:580247275435:layer:LambdaInsightsExtension:31
アジアパシフィック (大阪)	arn:aws:lambda:ap-northeast-3:194566237122:layer:LambdaInsightsExtension:2
アジアパシフィック (ソウル)	arn:aws:lambda:ap-northeast-2:580247275435:layer:LambdaInsightsExtension:32
アジアパシフィック (シンガポール)	arn:aws:lambda:ap-southeast-1:580247275435:layer:LambdaInsightsExtension:33
アジアパシフィック (シドニー)	arn:aws:lambda:ap-southeast-2:580247275435:layer:LambdaInsightsExtension:33
アジアパシフィック (東京)	arn:aws:lambda:ap-northeast-1:580247275435:layer:LambdaInsightsExtension:50
カナダ (中部)	arn:aws:lambda:ca-central-1:580247275435:layer:LambdaInsightsExtension:32
中国 (北京)	arn:aws-cn:lambda:cn-north-1:488211338238:layer:LambdaInsightsExtension:26
中国 (寧夏)	arn:aws-cn:lambda:cn-northwest-1:488211338238:layer:LambdaInsightsExtension:26
欧州 (フランクフルト)	arn:aws:lambda:eu-central-1:580247275435:layer:LambdaInsightsExtension:35
欧州 (アイルランド)	arn:aws:lambda:eu-west-1:580247275435:layer:LambdaInsightsExtension:33
欧州 (ロンドン)	arn:aws:lambda:eu-west-2:580247275435:layer:LambdaInsightsExtension:33

リージョン	ARN
欧州 (ミラノ)	arn:aws:lambda:eu-south-1:339249233099:layer:LambdaInsightsExtension:25
欧州 (パリ)	arn:aws:lambda:eu-west-3:580247275435:layer:LambdaInsightsExtension:32
欧州 (スペイン)	arn:aws:lambda:eu-south-2:352183217350:layer:LambdaInsightsExtension:10
欧州 (ストックホルム)	arn:aws:lambda:eu-north-1:580247275435:layer:LambdaInsightsExtension:30
欧州 (チューリッヒ)	arn:aws:lambda:eu-central-2:033019950311:layer:LambdaInsightsExtension:7
中東 (バーレーン)	arn:aws:lambda:me-south-1:285320876703:layer:LambdaInsightsExtension:25
中東 (アラブ首長国連邦)	arn:aws:lambda:me-central-1:732604637566:layer:LambdaInsightsExtension:9
南米 (サンパウロ)	arn:aws:lambda:sa-east-1:580247275435:layer:LambdaInsightsExtension:32

1.0.143.0

バージョン 1.0.143.0 には、Python 3.7 および Go 1.x との互換性に関するバグ修正が含まれています。Python 3.6 Lambda ランタイムは廃止されています。詳細については、「Lambda runtimes」(Lambda ランタイム) を参照してください。

バージョン 1.0.143.0 の ARN

次の表に、利用可能な各 AWS リージョンでこのバージョンの拡張機能で使用する ARN の一覧を示します。

リージョン	ARN
米国東部 (バージニア北部)	arn:aws:lambda:us-east-1:580247275435:layer:LambdaInsightsExtension:21
米国東部 (オハイオ)	arn:aws:lambda:us-east-2:580247275435:layer:LambdaInsightsExtension:21
米国西部 (北カリフォルニア)	arn:aws:lambda:us-west-1:580247275435:layer:LambdaInsightsExtension:20
米国西部 (オレゴン)	arn:aws:lambda:us-west-2:580247275435:layer:LambdaInsightsExtension:21
アフリカ (ケープタウン)	arn:aws:lambda:af-south-1:012438385374:layer:LambdaInsightsExtension:13
アジアパシフィック (香港)	arn:aws:lambda:ap-east-1:519774774795:layer:LambdaInsightsExtension:13
アジアパシフィック (ムンバイ)	arn:aws:lambda:ap-south-1:580247275435:layer:LambdaInsightsExtension:21
アジアパシフィック (大阪)	arn:aws:lambda:ap-northeast-3:194566237122:layer:LambdaInsightsExtension:2
アジアパシフィック (ソウル)	arn:aws:lambda:ap-northeast-2:580247275435:layer:LambdaInsightsExtension:20
アジアパシフィック (シンガポール)	arn:aws:lambda:ap-southeast-1:580247275435:layer:LambdaInsightsExtension:21
アジアパシフィック (シドニー)	arn:aws:lambda:ap-southeast-2:580247275435:layer:LambdaInsightsExtension:21
アジアパシフィック (東京)	arn:aws:lambda:ap-northeast-1:580247275435:layer:LambdaInsightsExtension:32

リージョン	ARN
カナダ (中部)	arn:aws:lambda:ca-central-1:580247275435:layer:LambdaInsightsExtension:20
中国 (北京)	arn:aws-cn:lambda:cn-north-1:488211338238:layer:LambdaInsightsExtension:14
中国 (寧夏)	arn:aws-cn:lambda:cn-northwest-1:488211338238:layer:LambdaInsightsExtension:14
欧州 (フランクフルト)	arn:aws:lambda:eu-central-1:580247275435:layer:LambdaInsightsExtension:21
欧州 (アイルランド)	arn:aws:lambda:eu-west-1:580247275435:layer:LambdaInsightsExtension:21
欧州 (ロンドン)	arn:aws:lambda:eu-west-2:580247275435:layer:LambdaInsightsExtension:21
欧州 (ミラノ)	arn:aws:lambda:eu-south-1:339249233099:layer:LambdaInsightsExtension:13
ヨーロッパ (パリ)	arn:aws:lambda:eu-west-3:580247275435:layer:LambdaInsightsExtension:20
ヨーロッパ (ストックホルム)	arn:aws:lambda:eu-north-1:580247275435:layer:LambdaInsightsExtension:20
中東 (バーレーン)	arn:aws:lambda:me-south-1:285320876703:layer:LambdaInsightsExtension:13
南米 (サンパウロ)	arn:aws:lambda:sa-east-1:580247275435:layer:LambdaInsightsExtension:20

1.0.135.0

バージョン 1.0.135.0 には、Lambda Insights でのディスクとファイル記述子の使用量の収集とレポートの方法に関するバグ修正が含まれています。拡張機能の以前のバージョンでは、関数を実行中

の /tmp ディレクトリの最大空き領域は tmp_free メトリクスによって報告されていました。このバージョンでは、代わりに最小値をレポートするようにメトリクスが変更されており、ディスク使用量を評価する上でさらに便利になっています。tmp ディレクトリのストレージクォータの詳細については、Lambda クォータを参照してください。

また、バージョン 1.0.135.0 では、オペレーティングシステムレベルではなく、プロセス全体の最大値としてファイル記述子の使用量 (fd_use および fd_max) が報告されるようになりました。

バージョン 1.0.135.0 の ARN

次の表に、利用可能な各 AWS リージョンでこのバージョンの拡張機能で使用する ARN の一覧を示します。

リージョン	ARN
米国東部 (バージニア北部)	arn:aws:lambda:us-east-1:580247275435:layer:LambdaInsightsExtension:18
米国東部 (オハイオ)	arn:aws:lambda:us-east-2:580247275435:layer:LambdaInsightsExtension:18
米国西部 (北カリフォルニア)	arn:aws:lambda:us-west-1:580247275435:layer:LambdaInsightsExtension:18
米国西部 (オレゴン)	arn:aws:lambda:us-west-2:580247275435:layer:LambdaInsightsExtension:18
アフリカ (ケープタウン)	arn:aws:lambda:af-south-1:012438385374:layer:LambdaInsightsExtension:11
アジアパシフィック (香港)	arn:aws:lambda:ap-east-1:519774774795:layer:LambdaInsightsExtension:11
アジアパシフィック (ムンバイ)	arn:aws:lambda:ap-south-1:580247275435:layer:LambdaInsightsExtension:18
アジアパシフィック (大阪)	arn:aws:lambda:ap-northeast-3:194566237122:layer:LambdaInsightsExtension:1

リージョン	ARN
アジアパシフィック (ソウル)	arn:aws:lambda:ap-northeast-2:580247275435:layer:LambdaInsightsExtension:18
アジアパシフィック (シンガポール)	arn:aws:lambda:ap-southeast-1:580247275435:layer:LambdaInsightsExtension:18
アジアパシフィック (シドニー)	arn:aws:lambda:ap-southeast-2:580247275435:layer:LambdaInsightsExtension:18
アジアパシフィック (東京)	arn:aws:lambda:ap-northeast-1:580247275435:layer:LambdaInsightsExtension:25
カナダ (中部)	arn:aws:lambda:ca-central-1:580247275435:layer:LambdaInsightsExtension:18
中国 (北京)	arn:aws-cn:lambda:cn-north-1:488211338238:layer:LambdaInsightsExtension:11
中国 (寧夏)	arn:aws-cn:lambda:cn-northwest-1:488211338238:layer:LambdaInsightsExtension:11
欧州 (フランクフルト)	arn:aws:lambda:eu-central-1:580247275435:layer:LambdaInsightsExtension:18
欧州 (アイルランド)	arn:aws:lambda:eu-west-1:580247275435:layer:LambdaInsightsExtension:18
欧州 (ロンドン)	arn:aws:lambda:eu-west-2:580247275435:layer:LambdaInsightsExtension:18
欧州 (ミラノ)	arn:aws:lambda:eu-south-1:339249233099:layer:LambdaInsightsExtension:11
ヨーロッパ (パリ)	arn:aws:lambda:eu-west-3:580247275435:layer:LambdaInsightsExtension:18
ヨーロッパ (ストックホルム)	arn:aws:lambda:eu-north-1:580247275435:layer:LambdaInsightsExtension:18

リージョン	ARN
中東 (バーレーン)	arn:aws:lambda:me-south-1:285320876703:layer:LambdaInsightsExtension:11
南米 (サンパウロ)	arn:aws:lambda:sa-east-1:580247275435:layer:LambdaInsightsExtension:18

1.0.119.0

バージョン 1.0.119.0 の ARN

次の表に、利用可能な各 AWS リージョンでこのバージョンの拡張機能で使用する ARN の一覧を示します。

リージョン	ARN
米国東部 (バージニア北部)	arn:aws:lambda:us-east-1:580247275435:layer:LambdaInsightsExtension:16
米国東部 (オハイオ)	arn:aws:lambda:us-east-2:580247275435:layer:LambdaInsightsExtension:16
米国西部 (北カリフォルニア)	arn:aws:lambda:us-west-1:580247275435:layer:LambdaInsightsExtension:16
米国西部 (オレゴン)	arn:aws:lambda:us-west-2:580247275435:layer:LambdaInsightsExtension:16
アフリカ (ケープタウン)	arn:aws:lambda:af-south-1:012438385374:layer:LambdaInsightsExtension:9
アジアパシフィック (香港)	arn:aws:lambda:ap-east-1:519774774795:layer:LambdaInsightsExtension:9
アジアパシフィック (ムンバイ)	arn:aws:lambda:ap-south-1:580247275435:layer:LambdaInsightsExtension:16

リージョン	ARN
アジアパシフィック (ソウル)	arn:aws:lambda:ap-northeast-2:580247275435:layer:LambdaInsightsExtension:16
アジアパシフィック (シンガポール)	arn:aws:lambda:ap-southeast-1:580247275435:layer:LambdaInsightsExtension:16
アジアパシフィック (シドニー)	arn:aws:lambda:ap-southeast-2:580247275435:layer:LambdaInsightsExtension:16
アジアパシフィック (東京)	arn:aws:lambda:ap-northeast-1:580247275435:layer:LambdaInsightsExtension:23
カナダ (中部)	arn:aws:lambda:ca-central-1:580247275435:layer:LambdaInsightsExtension:16
中国 (北京)	arn:aws-cn:lambda:cn-north-1:488211338238:layer:LambdaInsightsExtension:9
中国 (寧夏)	arn:aws-cn:lambda:cn-northwest-1:488211338238:layer:LambdaInsightsExtension:9
欧州 (フランクフルト)	arn:aws:lambda:eu-central-1:580247275435:layer:LambdaInsightsExtension:16
欧州 (アイルランド)	arn:aws:lambda:eu-west-1:580247275435:layer:LambdaInsightsExtension:16
欧州 (ロンドン)	arn:aws:lambda:eu-west-2:580247275435:layer:LambdaInsightsExtension:16
欧州 (ミラノ)	arn:aws:lambda:eu-south-1:339249233099:layer:LambdaInsightsExtension:9
ヨーロッパ (パリ)	arn:aws:lambda:eu-west-3:580247275435:layer:LambdaInsightsExtension:16
ヨーロッパ (ストックホルム)	arn:aws:lambda:eu-north-1:580247275435:layer:LambdaInsightsExtension:16

リージョン	ARN
中東 (バーレーン)	arn:aws:lambda:me-south-1:285320876703:layer:LambdaInsightsExtension:9
南米 (サンパウロ)	arn:aws:lambda:sa-east-1:580247275435:layer:LambdaInsightsExtension:16

1.0.98.0

このバージョンでは、不要なログが削除され、AWS Serverless Application Model CLI ローカルコールの問題も解決されています。この問題の詳細については、「Adding LambdaInsightsExtension results in timeout with 'sam local invoke'」をご参照ください。

バージョン 1.0.98.0 の ARN

次の表に、利用可能な各 AWS リージョンでこのバージョンの拡張機能で使用する ARN の一覧を示します。

リージョン	ARN
米国東部 (バージニア北部)	arn:aws:lambda:us-east-1:580247275435:layer:LambdaInsightsExtension:14
米国東部 (オハイオ)	arn:aws:lambda:us-east-2:580247275435:layer:LambdaInsightsExtension:14
米国西部 (北カリフォルニア)	arn:aws:lambda:us-west-1:580247275435:layer:LambdaInsightsExtension:14
米国西部 (オレゴン)	arn:aws:lambda:us-west-2:580247275435:layer:LambdaInsightsExtension:14
アフリカ (ケープタウン)	arn:aws:lambda:af-south-1:012438385374:layer:LambdaInsightsExtension:8
アジアパシフィック (香港)	arn:aws:lambda:ap-east-1:519774774795:layer:LambdaInsightsExtension:8

リージョン	ARN
アジアパシフィック (ムンバイ)	arn:aws:lambda:ap-south-1:580247275435:layer:LambdaInsightsExtension:14
アジアパシフィック (ソウル)	arn:aws:lambda:ap-northeast-2:580247275435:layer:LambdaInsightsExtension:14
アジアパシフィック (シンガポール)	arn:aws:lambda:ap-southeast-1:580247275435:layer:LambdaInsightsExtension:14
アジアパシフィック (シドニー)	arn:aws:lambda:ap-southeast-2:580247275435:layer:LambdaInsightsExtension:14
アジアパシフィック (東京)	arn:aws:lambda:ap-northeast-1:580247275435:layer:LambdaInsightsExtension:14
カナダ (中部)	arn:aws:lambda:ca-central-1:580247275435:layer:LambdaInsightsExtension:14
中国 (北京)	arn:aws-cn:lambda:cn-north-1:488211338238:layer:LambdaInsightsExtension:8
中国 (寧夏)	arn:aws-cn:lambda:cn-northwest-1:488211338238:layer:LambdaInsightsExtension:8
欧州 (フランクフルト)	arn:aws:lambda:eu-central-1:580247275435:layer:LambdaInsightsExtension:14
欧州 (アイルランド)	arn:aws:lambda:eu-west-1:580247275435:layer:LambdaInsightsExtension:14
欧州 (ロンドン)	arn:aws:lambda:eu-west-2:580247275435:layer:LambdaInsightsExtension:14
欧州 (ミラノ)	arn:aws:lambda:eu-south-1:339249233099:layer:LambdaInsightsExtension:8
ヨーロッパ (パリ)	arn:aws:lambda:eu-west-3:580247275435:layer:LambdaInsightsExtension:14

リージョン	ARN
ヨーロッパ (ストックホルム)	arn:aws:lambda:eu-north-1:580247275435:layer:LambdaInsightsExtension:14
中東 (バーレーン)	arn:aws:lambda:me-south-1:285320876703:layer:LambdaInsightsExtension:8
南米 (サンパウロ)	arn:aws:lambda:sa-east-1:580247275435:layer:LambdaInsightsExtension:14

1.0.89.0

このバージョンでは、常に関数の呼び出しの開始を表すように、パフォーマンスイベントタイムスタンプが修正されます。

バージョン 1.0.89.0 の ARN

次の表に、利用可能な各 AWS リージョンでこのバージョンの拡張機能で使用する ARN の一覧を示します。

リージョン	ARN
米国東部 (バージニア北部)	arn:aws:lambda:us-east-1:580247275435:layer:LambdaInsightsExtension:12
米国東部 (オハイオ)	arn:aws:lambda:us-east-2:580247275435:layer:LambdaInsightsExtension:12
米国西部 (北カリフォルニア)	arn:aws:lambda:us-west-1:580247275435:layer:LambdaInsightsExtension:12
米国西部 (オレゴン)	arn:aws:lambda:us-west-2:580247275435:layer:LambdaInsightsExtension:12
アジアパシフィック (ムンバイ)	arn:aws:lambda:ap-south-1:580247275435:layer:LambdaInsightsExtension:12

リージョン	ARN
アジアパシフィック (ソウル)	arn:aws:lambda:ap-northeast-2:580247275435:layer:LambdaInsightsExtension:12
アジアパシフィック (シンガポール)	arn:aws:lambda:ap-southeast-1:580247275435:layer:LambdaInsightsExtension:12
アジアパシフィック (シドニー)	arn:aws:lambda:ap-southeast-2:580247275435:layer:LambdaInsightsExtension:12
アジアパシフィック (東京)	arn:aws:lambda:ap-northeast-1:580247275435:layer:LambdaInsightsExtension:12
カナダ (中部)	arn:aws:lambda:ca-central-1:580247275435:layer:LambdaInsightsExtension:12
欧州 (フランクフルト)	arn:aws:lambda:eu-central-1:580247275435:layer:LambdaInsightsExtension:12
欧州 (アイルランド)	arn:aws:lambda:eu-west-1:580247275435:layer:LambdaInsightsExtension:12
欧州 (ロンドン)	arn:aws:lambda:eu-west-2:580247275435:layer:LambdaInsightsExtension:12
欧州 (パリ)	arn:aws:lambda:eu-west-3:580247275435:layer:LambdaInsightsExtension:12
欧州 (ストックホルム)	arn:aws:lambda:eu-north-1:580247275435:layer:LambdaInsightsExtension:12
南米 (サンパウロ)	arn:aws:lambda:sa-east-1:580247275435:layer:LambdaInsightsExtension:12

1.0.86.0

拡張機能のバージョン 1.0.54.0 では、メモリメトリクスが誤って報告され、100% より高くなることがありました。バージョン 1.0.86.0 では、Lambda プラットフォームメトリクスと同じイベント

データを使用して、メモリ測定の問題が修正されています。これは、記録されたメモリメトリクス値が劇的に変化することを意味します。これは、新しい Lambda Logs API を使用して実現されます。これにより、Lambda サンドボックスのメモリ使用量をより正確に測定できます。ただし、関数サンドボックスがタイムアウトし、その後スピンダウンされた場合、Lambda Logs API はプラットフォームレポートイベントを配信できないことに注意が必要です。この場合、Lambda Insights は呼び出しメトリクスを記録できません。Lambda Logs API の詳細については、「AWS Lambda Logs API」を参照してください。

バージョン 1.0.86.0 の新機能

- Lambda Logs API を使用してメモリメトリクスを修正します。これにより、メモリ統計が 100% を超えるという以前の問題が解決されました。
- 新しい CloudWatch メトリクスとして `Init Duration` をご紹介します。
- 呼び出し ARN を使用して、エイリアスおよび呼び出されたバージョンのバージョンディメンションを追加します。Lambda エイリアスまたはバージョンを使用して増分デプロイ (グリーン/ブルーデプロイなど) を実現する場合は、呼び出されたエイリアスに基づいてメトリクスを表示できます。関数でエイリアスまたはバージョンを使用しない場合、バージョンディメンションは適用されません。詳細については、「Lambda 関数エイリアス」を参照してください。
- パフォーマンスイベントに `billed_mb_ms field` を追加し、呼び出しあたりのコストを表示します。これは、プロビジョニングされた同時実行に関連するコストを考慮しません。
- パフォーマンスイベントに `billed_duration` および `duration` フィールドを追加します。

バージョン 1.0.86.0 の ARN

次の表に、利用可能な各 AWS リージョンでこのバージョンの拡張機能で使用する ARN の一覧を示します。

リージョン	ARN
米国東部 (バージニア北部)	`arn:aws:lambda:us-east-1:580247275435:layer:LambdaInsightsExtension:11`
米国東部 (オハイオ)	`arn:aws:lambda:us-east-2:580247275435:layer:LambdaInsightsExtension:11`
米国西部 (北カリフォルニア)	`arn:aws:lambda:us-west-1:580247275435:layer:LambdaInsightsExtension:11`

リージョン	ARN
米国西部 (オレゴン)	arn:aws:lambda:us-west-2:580247275435:layer:LambdaInsightsExtension:11
アジアパシフィック (ムンバイ)	arn:aws:lambda:ap-south-1:580247275435:layer:LambdaInsightsExtension:11
アジアパシフィック (ソウル)	arn:aws:lambda:ap-northeast-2:580247275435:layer:LambdaInsightsExtension:11
アジアパシフィック (シンガポール)	arn:aws:lambda:ap-southeast-1:580247275435:layer:LambdaInsightsExtension:11
アジアパシフィック (シドニー)	arn:aws:lambda:ap-southeast-2:580247275435:layer:LambdaInsightsExtension:11
アジアパシフィック (東京)	arn:aws:lambda:ap-northeast-1:580247275435:layer:LambdaInsightsExtension:11
カナダ (中部)	arn:aws:lambda:ca-central-1:580247275435:layer:LambdaInsightsExtension:11
欧州 (フランクフルト)	arn:aws:lambda:eu-central-1:580247275435:layer:LambdaInsightsExtension:11
欧州 (アイルランド)	arn:aws:lambda:eu-west-1:580247275435:layer:LambdaInsightsExtension:11
欧州 (ロンドン)	arn:aws:lambda:eu-west-2:580247275435:layer:LambdaInsightsExtension:11
欧州 (パリ)	arn:aws:lambda:eu-west-3:580247275435:layer:LambdaInsightsExtension:11
欧州 (ストックホルム)	arn:aws:lambda:eu-north-1:580247275435:layer:LambdaInsightsExtension:11
南米 (サンパウロ)	arn:aws:lambda:sa-east-1:580247275435:layer:LambdaInsightsExtension:11

1.0.54.0

バージョン 1.0.54.0 は、Lambda Insights 拡張機能の初回リリースでした。

バージョン 1.0.54.0 の ARN

次の表に、利用可能な各 AWS リージョンでこのバージョンの拡張機能で使用する ARN の一覧を示します。

リージョン	ARN
米国東部 (バージニア北部)	arn:aws:lambda:us-east-1:580247275435:layer:LambdaInsightsExtension:2
米国東部 (オハイオ)	arn:aws:lambda:us-east-2:580247275435:layer:LambdaInsightsExtension:2
米国西部 (北カリフォルニア)	arn:aws:lambda:us-west-1:580247275435:layer:LambdaInsightsExtension:2
米国西部 (オレゴン)	arn:aws:lambda:us-west-2:580247275435:layer:LambdaInsightsExtension:2
アジアパシフィック (ムンバイ)	arn:aws:lambda:ap-south-1:580247275435:layer:LambdaInsightsExtension:2
アジアパシフィック (ソウル)	arn:aws:lambda:ap-northeast-2:580247275435:layer:LambdaInsightsExtension:2
アジアパシフィック (シンガポール)	arn:aws:lambda:ap-southeast-1:580247275435:layer:LambdaInsightsExtension:2
アジアパシフィック (シドニー)	arn:aws:lambda:ap-southeast-2:580247275435:layer:LambdaInsightsExtension:2
アジアパシフィック (東京)	arn:aws:lambda:ap-northeast-1:580247275435:layer:LambdaInsightsExtension:2
カナダ (中部)	arn:aws:lambda:ca-central-1:580247275435:layer:LambdaInsightsExtension:2

リージョン	ARN
欧州 (フランクフルト)	arn:aws:lambda:eu-central-1:580247275435:layer:LambdaInsightsExtension:2
欧州 (アイルランド)	arn:aws:lambda:eu-west-1:580247275435:layer:LambdaInsightsExtension:2
欧州 (ロンドン)	arn:aws:lambda:eu-west-2:580247275435:layer:LambdaInsightsExtension:2
欧州 (パリ)	arn:aws:lambda:eu-west-3:580247275435:layer:LambdaInsightsExtension:2
欧州 (ストックホルム)	arn:aws:lambda:eu-north-1:580247275435:layer:LambdaInsightsExtension:2
南米 (サンパウロ)	arn:aws:lambda:sa-east-1:580247275435:layer:LambdaInsightsExtension:2

ARM64 プラットフォーム

このセクションでは、ARM64 プラットフォーム向け Lambda Insights 拡張機能のバージョンと、これらの拡張機能が各 AWS リージョンで使用する ARN の一覧を示します。

> **Important**
>
> Lambda Insights の拡張機能、1.0.317.0 以降は、Amazon Linux 1 をサポートしていません。

1.0.317.0

バージョン 1.0.317.0 には、Amazon Linux 1 プラットフォームのサポートの削除とバグ修正が含まれています。AWS GovCloud (US) リージョンのサポートも含まれています。

リージョン	ARN
米国東部 (バージニア北部)	arn:aws:lambda:us-east-1:580247275435:layer:LambdaInsightsExtension-Arm64:19

リージョン	ARN
米国東部 (オハイオ)	arn:aws:lambda:us-east-2:580247275435:layer:LambdaInsightsExtension-Arm64:21
米国西部 (北カリフォルニア)	arn:aws:lambda:us-west-1:580247275435:layer:LambdaInsightsExtension-Arm64:17
米国西部 (オレゴン)	arn:aws:lambda:us-west-2:580247275435:layer:LambdaInsightsExtension-Arm64:19
アフリカ (ケープタウン)	arn:aws:lambda:af-south-1:012438385374:layer:LambdaInsightsExtension-Arm64:17
アジアパシフィック (香港)	arn:aws:lambda:ap-east-1:519774774795:layer:LambdaInsightsExtension-Arm64:17
アジアパシフィック (ハイデラバード)	arn:aws:lambda:ap-south-2:891564319516:layer:LambdaInsightsExtension-Arm64:5
アジアパシフィック (ジャカルタ)	arn:aws:lambda:ap-southeast-3:439286490199:layer:LambdaInsightsExtension-Arm64:17
アジアパシフィック (ムンバイ)	arn:aws:lambda:ap-south-1:580247275435:layer:LambdaInsightsExtension-Arm64:21
アジアパシフィック (大阪)	arn:aws:lambda:ap-northeast-3:194566237122:layer:LambdaInsightsExtension-Arm64:16
アジアパシフィック (ソウル)	arn:aws:lambda:ap-northeast-2:580247275435:layer:LambdaInsightsExtension-Arm64:18
アジアパシフィック (シンガポール)	arn:aws:lambda:ap-southeast-1:580247275435:layer:LambdaInsightsExtension-Arm64:19
アジアパシフィック (シドニー)	arn:aws:lambda:ap-southeast-2:580247275435:layer:LambdaInsightsExtension-Arm64:19
アジアパシフィック (東京)	arn:aws:lambda:ap-northeast-1:580247275435:layer:LambdaInsightsExtension-Arm64:30

リージョン	ARN
カナダ (中部)	arn:aws:lambda:ca-central-1:580247275435:layer:LambdaInsightsExtension-Arm64:17
欧州 (フランクフルト)	arn:aws:lambda:eu-central-1:580247275435:layer:LambdaInsightsExtension-Arm64:19
欧州 (アイルランド)	arn:aws:lambda:eu-west-1:580247275435:layer:LambdaInsightsExtension-Arm64:19
欧州 (ロンドン)	arn:aws:lambda:eu-west-2:580247275435:layer:LambdaInsightsExtension-Arm64:19
欧州 (ミラノ)	arn:aws:lambda:eu-south-1:339249233099:layer:LambdaInsightsExtension-Arm64:17
欧州 (パリ)	arn:aws:lambda:eu-west-3:580247275435:layer:LambdaInsightsExtension-Arm64:17
欧州 (スペイン)	arn:aws:lambda:eu-south-2:352183217350:layer:LambdaInsightsExtension-Arm64:5
欧州 (ストックホルム)	arn:aws:lambda:eu-north-1:580247275435:layer:LambdaInsightsExtension-Arm64:17
中東 (バーレーン)	arn:aws:lambda:me-south-1:285320876703:layer:LambdaInsightsExtension-Arm64:17
南米 (サンパウロ)	arn:aws:lambda:sa-east-1:580247275435:layer:LambdaInsightsExtension-Arm64:17
AWS GovCloud (米国東部)	arn:aws-us-gov:lambda:us-gov-east-1:122132214140:layer:LambdaInsightsExtension-Arm64:1
AWS GovCloud (米国西部)	arn:aws-us-gov:lambda:us-gov-west-1:751350123760:layer:LambdaInsightsExtension-Arm64:1

1.0.295.0

バージョン 1.0.295.0 には、互換性のあるすべてのランタイム用の依存関係の更新が含まれています。

リージョン	ARN
米国東部 (バージニア北部)	arn:aws:lambda:us-east-1:580247275435:layer:LambdaInsightsExtension-Arm64:18
米国東部 (オハイオ)	arn:aws:lambda:us-east-2:580247275435:layer:LambdaInsightsExtension-Arm64:20
米国西部 (北カリフォルニア)	arn:aws:lambda:us-west-1:580247275435:layer:LambdaInsightsExtension-Arm64:16
米国西部 (オレゴン)	arn:aws:lambda:us-west-2:580247275435:layer:LambdaInsightsExtension-Arm64:18
アフリカ (ケープタウン)	arn:aws:lambda:af-south-1:012438385374:layer:LambdaInsightsExtension-Arm64:16
アジアパシフィック (香港)	arn:aws:lambda:ap-east-1:519774774795:layer:LambdaInsightsExtension-Arm64:16
アジアパシフィック (ハイデラバード)	arn:aws:lambda:ap-south-2:891564319516:layer:LambdaInsightsExtension-Arm64:4
アジアパシフィック (ジャカルタ)	arn:aws:lambda:ap-southeast-3:439286490199:layer:LambdaInsightsExtension-Arm64:16
アジアパシフィック (ムンバイ)	arn:aws:lambda:ap-south-1:580247275435:layer:LambdaInsightsExtension-Arm64:20
アジアパシフィック (大阪)	arn:aws:lambda:ap-northeast-3:194566237122:layer:LambdaInsightsExtension-Arm64:15
アジアパシフィック (ソウル)	arn:aws:lambda:ap-northeast-2:580247275435:layer:LambdaInsightsExtension-Arm64:17

リージョン	ARN
アジアパシフィック (シンガポール)	arn:aws:lambda:ap-southeast-1:580247275435:layer:LambdaInsightsExtension-Arm64:18
アジアパシフィック (シドニー)	arn:aws:lambda:ap-southeast-2:580247275435:layer:LambdaInsightsExtension-Arm64:18
アジアパシフィック (東京)	arn:aws:lambda:ap-northeast-1:580247275435:layer:LambdaInsightsExtension-Arm64:29
カナダ (中部)	arn:aws:lambda:ca-central-1:580247275435:layer:LambdaInsightsExtension-Arm64:16
欧州 (フランクフルト)	arn:aws:lambda:eu-central-1:580247275435:layer:LambdaInsightsExtension-Arm64:18
欧州 (アイルランド)	arn:aws:lambda:eu-west-1:580247275435:layer:LambdaInsightsExtension-Arm64:18
欧州 (ロンドン)	arn:aws:lambda:eu-west-2:580247275435:layer:LambdaInsightsExtension-Arm64:18
欧州 (ミラノ)	arn:aws:lambda:eu-south-1:339249233099:layer:LambdaInsightsExtension-Arm64:16
欧州 (パリ)	arn:aws:lambda:eu-west-3:580247275435:layer:LambdaInsightsExtension-Arm64:16
欧州 (スペイン)	arn:aws:lambda:eu-south-2:352183217350:layer:LambdaInsightsExtension-Arm64:4
欧州 (ストックホルム)	arn:aws:lambda:eu-north-1:580247275435:layer:LambdaInsightsExtension-Arm64:16
中東 (バーレーン)	arn:aws:lambda:me-south-1:285320876703:layer:LambdaInsightsExtension-Arm64:16
南米 (サンパウロ)	arn:aws:lambda:sa-east-1:580247275435:layer:LambdaInsightsExtension-Arm64:16

1.0.275.0

バージョン 1.0.275.0 には、互換性のあるすべてのランタイムのバグ修正と、欧州 (スペイン) およびアジアパシフィック (ハイデラバード) リージョンのサポートが含まれています。

リージョン	ARN
米国東部 (バージニア北部)	arn:aws:lambda:us-east-1:580247275435:layer:LambdaInsightsExtension-Arm64:16
米国東部 (オハイオ)	arn:aws:lambda:us-east-2:580247275435:layer:LambdaInsightsExtension-Arm64:18
米国西部 (北カリフォルニア)	arn:aws:lambda:us-west-1:580247275435:layer:LambdaInsightsExtension-Arm64:14
米国西部 (オレゴン)	arn:aws:lambda:us-west-2:580247275435:layer:LambdaInsightsExtension-Arm64:16
アフリカ (ケープタウン)	arn:aws:lambda:af-south-1:012438385374:layer:LambdaInsightsExtension-Arm64:14
アジアパシフィック (香港)	arn:aws:lambda:ap-east-1:519774774795:layer:LambdaInsightsExtension-Arm64:14
アジアパシフィック (ハイデラバード)	arn:aws:lambda:ap-south-2:891564319516:layer:LambdaInsightsExtension-Arm64:2
アジアパシフィック (ジャカルタ)	arn:aws:lambda:ap-southeast-3:439286490199:layer:LambdaInsightsExtension-Arm64:14
アジアパシフィック (ムンバイ)	arn:aws:lambda:ap-south-1:580247275435:layer:LambdaInsightsExtension-Arm64:18
アジアパシフィック (大阪)	arn:aws:lambda:ap-northeast-3:194566237122:layer:LambdaInsightsExtension-Arm64:13
アジアパシフィック (ソウル)	arn:aws:lambda:ap-northeast-2:580247275435:layer:LambdaInsightsExtension-Arm64:15

リージョン	ARN
アジアパシフィック (シンガポール)	arn:aws:lambda:ap-southeast-1:580247275435:layer:LambdaInsightsExtension-Arm64:16
アジアパシフィック (シドニー)	arn:aws:lambda:ap-southeast-2:580247275435:layer:LambdaInsightsExtension-Arm64:16
アジアパシフィック (東京)	arn:aws:lambda:ap-northeast-1:580247275435:layer:LambdaInsightsExtension-Arm64:27
カナダ (中部)	arn:aws:lambda:ca-central-1:580247275435:layer:LambdaInsightsExtension-Arm64:14
欧州 (フランクフルト)	arn:aws:lambda:eu-central-1:580247275435:layer:LambdaInsightsExtension-Arm64:16
欧州 (アイルランド)	arn:aws:lambda:eu-west-1:580247275435:layer:LambdaInsightsExtension-Arm64:16
欧州 (ロンドン)	arn:aws:lambda:eu-west-2:580247275435:layer:LambdaInsightsExtension-Arm64:16
欧州 (ミラノ)	arn:aws:lambda:eu-south-1:339249233099:layer:LambdaInsightsExtension-Arm64:14
欧州 (パリ)	arn:aws:lambda:eu-west-3:580247275435:layer:LambdaInsightsExtension-Arm64:14
欧州 (スペイン)	arn:aws:lambda:eu-south-2:352183217350:layer:LambdaInsightsExtension-Arm64:2
欧州 (ストックホルム)	arn:aws:lambda:eu-north-1:580247275435:layer:LambdaInsightsExtension-Arm64:14
中東 (バーレーン)	arn:aws:lambda:me-south-1:285320876703:layer:LambdaInsightsExtension-Arm64:14
南米 (サンパウロ)	arn:aws:lambda:sa-east-1:580247275435:layer:LambdaInsightsExtension-Arm64:14

1.0.273.0

バージョン 1.0.273.0 には、互換性のあるすべてのランタイム用のバグ修正が含まれています。

リージョン	ARN
米国東部 (バージニア北部)	arn:aws:lambda:us-east-1:580247275435:layer:LambdaInsightsExtension-Arm64:12
米国東部 (オハイオ)	arn:aws:lambda:us-east-2:580247275435:layer:LambdaInsightsExtension-Arm64:14
米国西部 (北カリフォルニア)	arn:aws:lambda:us-west-1:580247275435:layer:LambdaInsightsExtension-Arm64:9
米国西部 (オレゴン)	arn:aws:lambda:us-west-2:580247275435:layer:LambdaInsightsExtension-Arm64:12
アフリカ (ケープタウン)	arn:aws:lambda:af-south-1:012438385374:layer:LambdaInsightsExtension-Arm64:9
アジアパシフィック (香港)	arn:aws:lambda:ap-east-1:519774774795:layer:LambdaInsightsExtension-Arm64:9
アジアパシフィック (ジャカルタ)	arn:aws:lambda:ap-southeast-3:439286490199:layer:LambdaInsightsExtension-Arm64:9
アジアパシフィック (ムンバイ)	arn:aws:lambda:ap-south-1:580247275435:layer:LambdaInsightsExtension-Arm64:14
アジアパシフィック (大阪)	arn:aws:lambda:ap-northeast-3:194566237122:layer:LambdaInsightsExtension-Arm64:9
アジアパシフィック (ソウル)	arn:aws:lambda:ap-northeast-2:580247275435:layer:LambdaInsightsExtension-Arm64:11
アジアパシフィック (シンガポール)	arn:aws:lambda:ap-southeast-1:580247275435:layer:LambdaInsightsExtension-Arm64:12

リージョン	ARN
アジアパシフィック (シドニー)	arn:aws:lambda:ap-southeast-2:580247275435:layer:LambdaInsightsExtension-Arm64:12
アジアパシフィック (東京)	arn:aws:lambda:ap-northeast-1:580247275435:layer:LambdaInsightsExtension-Arm64:23
カナダ (中部)	arn:aws:lambda:ca-central-1:580247275435:layer:LambdaInsightsExtension-Arm64:10
欧州 (フランクフルト)	arn:aws:lambda:eu-central-1:580247275435:layer:LambdaInsightsExtension-Arm64:12
欧州 (アイルランド)	arn:aws:lambda:eu-west-1:580247275435:layer:LambdaInsightsExtension-Arm64:12
欧州 (ロンドン)	arn:aws:lambda:eu-west-2:580247275435:layer:LambdaInsightsExtension-Arm64:12
欧州 (ミラノ)	arn:aws:lambda:eu-south-1:339249233099:layer:LambdaInsightsExtension-Arm64:9
ヨーロッパ (パリ)	arn:aws:lambda:eu-west-3:580247275435:layer:LambdaInsightsExtension-Arm64:10
ヨーロッパ (ストックホルム)	arn:aws:lambda:eu-north-1:580247275435:layer:LambdaInsightsExtension-Arm64:10
中東 (バーレーン)	arn:aws:lambda:me-south-1:285320876703:layer:LambdaInsightsExtension-Arm64:9
南米 (サンパウロ)	arn:aws:lambda:sa-east-1:580247275435:layer:LambdaInsightsExtension-Arm64:10

1.0.229.0

バージョン 1.0.229.0 には、互換性のあるすべてのランタイム用のバグ修正が含まれています。また、次のリージョン用のサポートも追加されます。

- 米国西部 (北カリフォルニア)
- アフリカ (ケープタウン)
- アジアパシフィック (香港)
- アジアパシフィック (ジャカルタ)
- アジアパシフィック (大阪)
- アジアパシフィック (ソウル)
- カナダ (中部)
- 欧州 (ミラノ)
- ヨーロッパ (パリ)
- ヨーロッパ (ストックホルム)
- 中東 (バーレーン)
- 南米 (サンパウロ)

リージョン	ARN
米国東部 (バージニア北部)	arn:aws:lambda:us-east-1:580247275435:layer:LambdaInsightsExtension-Arm64:5
米国東部 (オハイオ)	arn:aws:lambda:us-east-2:580247275435:layer:LambdaInsightsExtension-Arm64:7
米国西部 (北カリフォルニア)	arn:aws:lambda:us-west-1:580247275435:layer:LambdaInsightsExtension-Arm64:3
米国西部 (オレゴン)	arn:aws:lambda:us-west-2:580247275435:layer:LambdaInsightsExtension-Arm64:5
アフリカ (ケープタウン)	arn:aws:lambda:af-south-1:012438385374:layer:LambdaInsightsExtension-Arm64:2
アジアパシフィック (香港)	arn:aws:lambda:ap-east-1:519774774795:layer:LambdaInsightsExtension-Arm64:2
アジアパシフィック (ジャカルタ)	arn:aws:lambda:ap-southeast-3:439286490199:layer:LambdaInsightsExtension-Arm64:2

リージョン	ARN
アジアパシフィック（ムンバイ）	arn:aws:lambda:ap-south-1:580247275435:layer:LambdaInsightsExtension-Arm64:7
アジアパシフィック (大阪)	arn:aws:lambda:ap-northeast-3:194566237122:layer:LambdaInsightsExtension-Arm64:2
アジアパシフィック (ソウル)	arn:aws:lambda:ap-northeast-2:580247275435:layer:LambdaInsightsExtension-Arm64:4
アジアパシフィック (シンガポール)	arn:aws:lambda:ap-southeast-1:580247275435:layer:LambdaInsightsExtension-Arm64:5
アジアパシフィック (シドニー)	arn:aws:lambda:ap-southeast-2:580247275435:layer:LambdaInsightsExtension-Arm64:5
アジアパシフィック (東京)	arn:aws:lambda:ap-northeast-1:580247275435:layer:LambdaInsightsExtension-Arm64:11
カナダ (中部)	arn:aws:lambda:ca-central-1:580247275435:layer:LambdaInsightsExtension-Arm64:3
欧州 (フランクフルト)	arn:aws:lambda:eu-central-1:580247275435:layer:LambdaInsightsExtension-Arm64:5
欧州 (アイルランド)	arn:aws:lambda:eu-west-1:580247275435:layer:LambdaInsightsExtension-Arm64:5
欧州 (ロンドン)	arn:aws:lambda:eu-west-2:580247275435:layer:LambdaInsightsExtension-Arm64:5
欧州 (スペイン)	arn:aws:lambda:eu-south-1:339249233099:layer:LambdaInsightsExtension-Arm64:2
欧州 (パリ)	arn:aws:lambda:eu-west-3:580247275435:layer:LambdaInsightsExtension-Arm64:3
ヨーロッパ (ストックホルム)	arn:aws:lambda:eu-north-1:580247275435:layer:LambdaInsightsExtension-Arm64:3

リージョン	ARN
中東 (バーレーン)	arn:aws:lambda:me-south-1:285320876703:layer:LambdaInsightsExtension-Arm64:2
南米 (サンパウロ)	arn:aws:lambda:sa-east-1:580247275435:layer:LambdaInsightsExtension-Arm64:3

1.0.135.0

バージョン 1.0.135.0 には、Lambda Insights でのディスクとファイル記述子の使用量の収集とレポートの方法に関するバグ修正が含まれています。拡張機能の以前のバージョンでは、関数を実行中の /tmp ディレクトリの最大空き領域は tmp_free メトリクスによって報告されていました。このバージョンでは、代わりに最小値をレポートするようにメトリクスが変更されており、ディスク使用量を評価する上でさらに便利になっています。tmp ディレクトリのストレージクォータの詳細については、Lambda クォータを参照してください。

また、バージョン 1.0.135.0 では、オペレーティングシステムレベルではなく、プロセス全体の最大値としてファイル記述子の使用量 (fd_use および fd_max) が報告されるようになりました。

リージョン	ARN
米国東部 (バージニア北部)	arn:aws:lambda:us-east-1:580247275435:layer:LambdaInsightsExtension-Arm64:2
米国東部 (オハイオ)	arn:aws:lambda:us-east-2:580247275435:layer:LambdaInsightsExtension-Arm64:2
米国西部 (オレゴン)	arn:aws:lambda:us-west-2:580247275435:layer:LambdaInsightsExtension-Arm64:2
アジアパシフィック (ムンバイ)	arn:aws:lambda:ap-south-1:580247275435:layer:LambdaInsightsExtension-Arm64:2
アジアパシフィック (シンガポール)	arn:aws:lambda:ap-southeast-1:580247275435:layer:LambdaInsightsExtension-Arm64:2

リージョン	ARN
アジアパシフィック (シドニー)	arn:aws:lambda:ap-southeast-2:580247275435:layer:LambdaInsightsExtension-Arm64:2
アジアパシフィック (東京)	arn:aws:lambda:ap-northeast-1:580247275435:layer:LambdaInsightsExtension-Arm64:2
欧州 (フランクフルト)	arn:aws:lambda:eu-central-1:580247275435:layer:LambdaInsightsExtension-Arm64:2
欧州 (アイルランド)	arn:aws:lambda:eu-west-1:580247275435:layer:LambdaInsightsExtension-Arm64:2
欧州 (ロンドン)	arn:aws:lambda:eu-west-2:580247275435:layer:LambdaInsightsExtension-Arm64:2

1.0.119.0

リージョン	ARN
米国東部 (バージニア北部)	arn:aws:lambda:us-east-1:580247275435:layer:LambdaInsightsExtension-Arm64:1
米国東部 (オハイオ)	arn:aws:lambda:us-east-2:580247275435:layer:LambdaInsightsExtension-Arm64:1
米国西部 (オレゴン)	arn:aws:lambda:us-west-2:580247275435:layer:LambdaInsightsExtension-Arm64:1
アジアパシフィック (ムンバイ)	arn:aws:lambda:ap-south-1:580247275435:layer:LambdaInsightsExtension-Arm64:1
アジアパシフィック (シンガポール)	arn:aws:lambda:ap-southeast-1:580247275435:layer:LambdaInsightsExtension-Arm64:1
アジアパシフィック (シドニー)	arn:aws:lambda:ap-southeast-2:580247275435:layer:LambdaInsightsExtension-Arm64:1

リージョン	ARN
アジアパシフィック (東京)	arn:aws:lambda:ap-northeast-1:580247275435:layer:LambdaInsightsExtension-Arm64:1
欧州 (フランクフルト)	arn:aws:lambda:eu-central-1:580247275435:layer:LambdaInsightsExtension-Arm64:1
欧州 (アイルランド)	arn:aws:lambda:eu-west-1:580247275435:layer:LambdaInsightsExtension-Arm64:1
欧州 (ロンドン)	arn:aws:lambda:eu-west-2:580247275435:layer:LambdaInsightsExtension-Arm64:1

コンソールを使用して既存の Lambda 関数で Lambda Insights を有効にする

Lambda コンソールで次の手順を使用して、既存の Lambda 関数で Lambda Insights を有効にします。

Lambda 関数で Lambda Insights を有効にするには

1. AWS Lambda コンソールを https://console.aws.amazon.com/lambda/ で開きます。
2. 関数の名前を選択し、次の画面で [Configuration] (設定) タブを選択します。
3. [設定] タブで、左側のナビゲーションメニューから [モニタリングおよび運用ツール] を選択し、[編集] を選択します。

   モニターリングツールを編集できる画面に移動します。
4. [Lambda Insights 拡張モニタリング] で、[編集] を選択します。
5. [CloudWatch Lambda インサイト] のセクションで [拡張モニタリング] を有効にして、[保存] を選択します。

既存の Lambda 関数で Lambda Insights を有効にするために AWS CLI を使用する

AWS CLI を使用して、既存の Lambda 関数で Lambda Insights を有効にするには、次の手順に従います。

ステップ 1: 関数のアクセス許可を更新する

関数のアクセス許可を更新するには

- 次のコマンドを入力して、CloudWatchLambdaInsightsExecutionRolePolicy マネージド IAM ポリシーを関数の実行ロールにアタッチします。

```
aws iam attach-role-policy \
--role-name function-execution-role \
--policy-arn "arn:aws:iam::aws:policy/CloudWatchLambdaInsightsExecutionRolePolicy"
```

ステップ 2: Lambda 拡張機能をインストールする

Lambda 拡張機能をインストールするには、次のコマンドを入力します。`layers` パラメータの ARN 値を、使用するリージョンと拡張バージョンに一致する ARN に置き換えます。詳細については、「Lambda Insights 拡張機能の利用可能なバージョン」を参照してください。

```
aws lambda update-function-configuration \
   --function-name function-name \
   --layers "arn:aws:lambda:us-west-1:580247275435:layer:LambdaInsightsExtension:14"
```

ステップ 3: CloudWatch Logs VPC エンドポイントを有効にする

このステップは、インターネットにアクセスできないプライベートサブネットで実行されている関数、および CloudWatch Logs Virtual Private Cloud (VPC) エンドポイントをまだ設定していない場合にのみ必要です。

このステップを実行する必要がある場合は、次のコマンドを入力し、プレースホルダーを VPC の情報に置き換えます。

詳細については、「インターフェイス VPC エンドポイントでの CloudWatch Logs の使用」を参照してください。

```
aws ec2 create-vpc-endpoint \
--vpc-id vpcId \
--vpc-endpoint-type Interface \
--service-name com.amazonaws.region.logs \
--subnet-id subnetId
--security-group-id securitygroupId
```

既存の Lambda 関数で Lambda Insights を有効にするために AWS SAM CLI を使用する

AWS SAM AWS CLI を使用して、既存の Lambda 関数で Lambda Insights を有効にするには、次の手順に従います。

最新バージョンの AWS SAM CLI をまだインストールしていない場合は、まずインストールまたはアップグレードする必要があります。詳細については、AWS SAM CLI のインストールを参照してください。

ステップ 1: レイヤーをインストールする

Lambda Insights 拡張機能をすべての Lambda 関数で使用できるようにするには、Lambda Insights レイヤーの ARN を使用して SAM テンプレートの `Globals` セクションに `Layers` プロパティを追加します。次の例では、Lambda Insights の最初のリリースにレイヤーを使用しています。Lambda Insights 拡張レイヤーの最新リリースバージョンについては、「Lambda Insights 拡張機能の利用可能なバージョン」を参照してください。

```
Globals:
  Function:
    Layers:
      - !Sub "arn:aws:lambda:
${AWS::Region}:580247275435:layer:LambdaInsightsExtension:14"
```

このレイヤーを 1 つの関数に対してのみ有効にするには、この例のように関数に `Layers` プロパティを追加します。

```
Resources:
  MyFunction:
    Type: AWS::Serverless::Function
    Properties:
      Layers:
        - !Sub "arn:aws:lambda:
${AWS::Region}:580247275435:layer:LambdaInsightsExtension:14"
```

ステップ 2: マネージドポリシーを追加する

各関数について、CloudWatchLambdaInsightsExecutionRolePolicy IAM ポリシーを追加します。

AWS SAM はグローバルポリシーをサポートしていないため、この例に示すように、各関数で個別に有効にする必要があります。グローバルの詳細については、「グローバルセクション」を参照してください。

```
Resources:
  MyFunction:
    Type: AWS::Serverless::Function
    Properties:
      Policies:
        - CloudWatchLambdaInsightsExecutionRolePolicy
```

ローカルでの呼び出し

AWS SAM CLI は Lambda 拡張機能をサポートしています。ただし、ローカルで実行されるすべての呼び出しは、ランタイム環境をリセットします。ランタイムはシャットダウンイベントなしで再起動されるため、Lambda Insights データはローカル呼び出しからは使用できません。詳細については、「リリース 1.6.0 - AWS Lambda 拡張機能のローカルテストのサポートを追加する」を参照してください。

トラブルシューティング

Lambda Insights のインストールに関するトラブルシューティングを行うには、Lambda 関数に次の環境変数を追加して、デバッグログを有効にします。

```
Resources:
  MyFunction:
    Type: AWS::Serverless::Function
    Properties:
      Environment:
        Variables:
          LAMBDA_INSIGHTS_LOG_LEVEL: info
```

既存の Lambda 関数で Lambda Insights を有効にするために AWS CloudFormation を使用する

AWS CloudFormation を使用して、既存の Lambda 関数で Lambda Insights を有効にするには、次の手順に従います。

ステップ 1: レイヤーをインストールする

Lambda Insights レイヤーを Lambda Insights レイヤー ARN 内の `Layers` プロパティに追加します。次の例では、Lambda Insights の最初のリリースにレイヤーを使用しています。Lambda Insights 拡張レイヤーの最新リリースバージョンについては、「Lambda Insights 拡張機能の利用可能なバージョン」を参照してください。

```
Resources:
  MyFunction:
    Type: AWS::Lambda::Function
    Properties:
      Layers:
        - !Sub "arn:aws:lambda:
${AWS::Region}:580247275435:layer:LambdaInsightsExtension:14"
```

ステップ 2: マネージドポリシーを追加する

CloudWatchLambdaInsightsExecutionRolePolicy IAM ポリシーを関数実行ロールに追加します。

```
Resources:
  MyFunctionExecutionRole:
    Type: 'AWS::IAM::Role'
    Properties:
      ManagedPolicyArns:
        - 'arn:aws:iam::aws:policy/CloudWatchLambdaInsightsExecutionRolePolicy'
```

ステップ 3: (オプション) VPC エンドポイントを追加する

このステップは、インターネットにアクセスできないプライベートサブネットで実行されている関数、および CloudWatch Logs Virtual Private Cloud (VPC) エンドポイントをまだ設定していない場合にのみ必要です。詳細については、「インターフェイス VPC エンドポイントでの CloudWatch Logs の使用」を参照してください。

```
Resources:
  CloudWatchLogsVpcPrivateEndpoint:
    Type: AWS::EC2::VPCEndpoint
    Properties:
      PrivateDnsEnabled: 'true'
      VpcEndpointType: Interface
      VpcId: !Ref: VPC
      ServiceName: !Sub com.amazonaws.${AWS::Region}.logs
      SecurityGroupIds:
        - !Ref InterfaceVpcEndpointSecurityGroup
```

```
      SubnetIds:
        - !Ref PublicSubnet01
        - !Ref PublicSubnet02
        - !Ref PublicSubnet03
```

既存の Lambda 関数で Lambda Insights を有効にするために AWS CDK を使用する

AWS CDK を使用して、既存の Lambda 関数で Lambda Insights を有効にするには、次の手順に従います。これらのステップを使用するには、既に AWS CDK を使用してリソースを管理している必要があります。

このセクションのコマンドは、TypeScript で記載されています。

まず、関数のアクセス許可を更新します。

```
executionRole.addManagedPolicy(
 ManagedPolicy.fromAwsManagedPolicyName('CloudWatchLambdaInsightsExecutionRolePolicy')
);
```

次に、Lambda 関数に拡張機能をインストールします。`layerArn` パラメータの ARN 値を、使用するリージョンと拡張バージョンに一致する ARN に置き換えます。詳細については、「Lambda Insights 拡張機能の利用可能なバージョン」を参照してください。

```
import lambda = require('@aws-cdk/aws-lambda');
const layerArn = 'arn:aws:lambda:us-
west-1:580247275435:layer:LambdaInsightsExtension:14';
const layer = lambda.LayerVersion.fromLayerVersionArn(this, 'LayerFromArn', layerArn);
```

必要に応じて、CloudWatch Logs の Virtual Private Cloud (VPC) エンドポイントを使用することもできます。このステップは、インターネットにアクセスできないプライベートサブネットで実行されている関数、および CloudWatch Logs VPC エンドポイントをまだ設定していない場合にのみ必要です。

```
const cloudWatchLogsEndpoint = vpc.addInterfaceEndpoint('cwl-gateway', {
  service: InterfaceVpcEndpointAwsService.CLOUDWATCH_LOGS,
});

cloudWatchLogsEndpoint.connections.allowDefaultPortFromAnyIpv4();
```

Serverless Framework を使用して、既存の Lambda 関数で Lambda Insights を有効にする

Serverless Framework を使用して、既存の Lambda 関数で Lambda Insights を有効にするには、次の手順に従います。Serverless Framework の詳細については、serverless.com を参照してください。

これは、Serverless の Lambda Insights プラグインから行えます。詳細については、「serverless-plugin-lambda-insights」を参照してください。

最新バージョンの Serverless コマンドラインインターフェースをまだ使用していない場合は、まずインストールまたはアップグレードする必要があります。詳細については、Serverless Framework Open Source & AWS の使用開始を参照してください。

Serverless Framework を使用して、Lambda 関数で Lambda Insights を有効化するには

1. Serverless のディレクトリで次のコマンドを実行し、Lambda Insights 用の Serverless プラグインをインストールします。

```
npm install --save-dev serverless-plugin-lambda-insights
```

2. `serverless.yml`ファイルで、図のように `plugins` セクション内にプラグインを追加します。

```
provider:
  name: aws
plugins:
  - serverless-plugin-lambda-insights
```

3. Lambda Insights を有効にします。
   - 次のプロパティを serverless.yml ファイルに追加することで、関数ごとに個別に Lambda Insights を有効化できます。

```
functions:
  myLambdaFunction:
    handler: src/app/index.handler
    lambdaInsights: true #enables Lambda Insights for this function
```

   - 次のカスタムセクションを追加すると、`serverless.yml` ファイル内にあるすべての関数に対して Lambda Insights を有効化できます。

```
custom:
  lambdaInsights:
    defaultLambdaInsights: true #enables Lambda Insights for all functions
```

4. 次のコマンドを入力して、Serverless サービスを再デプロイします。

```
serverless deploy
```

これにより、すべての関数が再デプロイされ、指定した関数に対し Lambda Insights が有効になります。ここでは、Lambda Insights レイヤーを追加し、`arn:aws:iam::aws:policy/CloudWatchLambdaInsightsExecutionRolePolicy` IAM ポリシーを使用して必要なアクセス許可をアタッチすることで、Lambda Insights を有効にしています。

Lambda コンテナイメージをデプロイして Lambda Insights を有効化する

Dockerfile に行を追加し、コンテナイメージとしてデプロイされた Lambda 関数で Lambda Insights を有効にします。これらの行により、コンテナイメージに対し、Lambda Insights エージェントが拡張機能としてインストールされます。追加する行は、x86-64 コンテナと ARM64 コンテナで異なります。

> **Note**
>
> Lambda Insights エージェントは、Amazon Linux 2 を使用する Lambda ランタイムでのみサポートされます。

トピック

- x86-64 コンテナイメージのデプロイ
- ARM64 コンテナイメージのデプロイ

x86-64 コンテナイメージのデプロイ

x86-64 コンテナ上で実行されるコンテナイメージとしてデプロイされた Lambda 関数で Lambda Insights を有効にするには、Dockerfile に次の行を追加します。これらの行により、コンテナイメージに対し、Lambda Insights エージェントが拡張機能としてインストールされます。

```
RUN curl -O https://lambda-insights-extension.s3-ap-northeast-1.amazonaws.com/
amazon_linux/lambda-insights-extension.rpm && \
    rpm -U lambda-insights-extension.rpm && \
    rm -f lambda-insights-extension.rpm
```

Lambda 関数の作成後、CloudWatchLambdaInsightsExecutionRolePolicy IAM ポリシーを関数の実行ロールに割り当てると、コンテナイメージベースの Lambda 関数で Lambda Insights が有効化されます。

> **Note**
>
> Lambda Insights の拡張機能の古いバージョンを使用するには、前出のコマンドの URL を `https://lambda-insights-extension.s3-ap-northeast-1.amazonaws.com/amazon_linux/lambda-insights-extension.1.0.111.0.rpm` に置き換えます。現時点では、Lambda Insights のバージョン 1.0.111.0 以降のみが利用可能です。詳細については、「Lambda Insights 拡張機能の利用可能なバージョン」を参照してください。

Linux サーバーで Lambda Insights エージェントパッケージの署名を確認するには

1. 次のコマンドを入力して、公開キーをダウンロードします。

   ```
   shell$ wget https://lambda-insights-extension.s3-ap-northeast-1.amazonaws.com/
   lambda-insights-extension.gpg
   ```

2. 次のコマンドを入力して、公開鍵をキーリングにインポートします。

   ```
   shell$ gpg --import lambda-insights-extension.gpg
   ```

   出力は以下のようになります。次のステップで必要になるため、key 値を書きとめておきます。この出力例では、キー値は 848ABDC8 です。

   ```
   gpg: key 848ABDC8: public key "Amazon Lambda Insights Extension" imported
   gpg: Total number processed: 1
   gpg: imported: 1  (RSA: 1)
   ```

3. 次のコマンドを入力して、フィンガープリントを検証します。key-value の部分は、前の手順で記録したキーの値に置き換えます。

```
shell$  gpg --fingerprint key-value
```

このコマンドの出力では、フィンガープリント文字列は E0AF FA11 FFF3 5BD7 349E E222 479C 97A1 848A BDC8 のようになります。フィンガープリント文字列が一致しない場合は、エージェントのインストールを中止し、AWS にお問い合わせください。

4. フィンガープリントが確認できたら、それを使用して、Lambda Insights のエージェントパッケージを確認できます。次のコマンドを入力して、パッケージの署名ファイルをダウンロードします。

```
shell$  wget https://lambda-insights-extension.s3-ap-northeast-1.amazonaws.com/
amazon_linux/lambda-insights-extension.rpm.sig
```

5. 次のコマンドを入力して、署名を確認します。

```
shell$ gpg --verify lambda-insights-extension.rpm.sig lambda-insights-extension.rpm
```

出力は次のようになります。

```
gpg: Signature made Thu 08 Apr 2021 06:41:00 PM UTC using RSA key ID 848ABDC8
gpg: Good signature from "Amazon Lambda Insights Extension"
gpg: WARNING: This key is not certified with a trusted signature!
gpg:          There is no indication that the signature belongs to the owner.
Primary key fingerprint: E0AF FA11 FFF3 5BD7 349E  E222 479C 97A1 848A BDC8
```

このときの出力には、信頼された署名に関する警告が表示される場合があります。キーは、自分や信頼する人が署名した場合にのみ信頼できます。つまり、署名が無効であるわけではなく、パブリックキーを確認していないことになります。

出力に BAD signature が含まれている場合は、ここまでの手順を正しく実行したかどうかを確認します。この BAD signature レスポンスが引き続き出力される場合は、AWS に連絡の上、ダウンロードしたファイルの使用は中止してください。

x86-64 の例

このセクションでは、コンテナイメージベースの Python Lambda 関数で Lambda Insights を有効にする例を示します。

Lambda コンテナイメージで Lambda Insights を有効にする例

1. 次のような Dockerfile を作成します。

```
FROM public.ecr.aws/lambda/python:3.8

// extra lines to install the agent here
RUN curl -O https://lambda-insights-extension.s3-ap-northeast-1.amazonaws.com/
amazon_linux/lambda-insights-extension.rpm && \
    rpm -U lambda-insights-extension.rpm && \
    rm -f lambda-insights-extension.rpm

COPY index.py ${LAMBDA_TASK_ROOT}
CMD [ "index.handler" ]
```

2. 次のような Python ファイルを、index.py という名前で作成します。

```
def handler(event, context):
  return {
    'message': 'Hello World!'
  }
```

3. Dockerfile と index.py を同じディレクトリに保存します。次に、保存先のディレクトリで次の手順を実行して Docker イメージをビルドし、それを Amazon ECR にアップロードします。

```
// create an ECR repository
aws ecr create-repository --repository-name test-repository
// build the docker image
docker build -t test-image .
// sign in to AWS
aws ecr get-login-password | docker login --username AWS --password-stdin
 "${ACCOUNT_ID}".dkr.ecr."${REGION}".amazonaws.com
// tag the image
docker tag test-image:latest "${ACCOUNT_ID}".dkr.ecr."${REGION}".amazonaws.com/
test-repository:latest
// push the image to ECR
docker push "${ACCOUNT_ID}".dkr.ecr."${REGION}".amazonaws.com/test-
repository:latest
```

4. 先ほど作成した Amazon ECR イメージを使用して、Lambda 関数を作成します。
5. CloudWatchLambdaInsightsExecutionRolePolicy IAM ポリシーを、関数の実行ロールに割り当てます。

ARM64 コンテナイメージのデプロイ

AL2_aarch64 コンテナ (ARM64 アーキテクチャを使用) 上で実行されるコンテナイメージとしてデプロイされた Lambda 関数で Lambda Insights を有効にするには、Dockerfile に次の行を追加します。これらの行により、コンテナイメージに対し、Lambda Insights エージェントが拡張機能としてインストールされます。

```
RUN curl -O https://lambda-insights-extension-arm64.s3-ap-northeast-1.amazonaws.com/
amazon_linux/lambda-insights-extension-arm64.rpm && \
    rpm -U lambda-insights-extension-arm64.rpm && \
    rm -f lambda-insights-extension-arm64.rpm
```

Lambda 関数の作成後、CloudWatchLambdaInsightsExecutionRolePolicy IAM ポリシーを関数の実行ロールに割り当てると、コンテナイメージベースの Lambda 関数で Lambda Insights が有効化されます。

> **Note**
>
> Lambda Insights の拡張機能の古いバージョンを使用するには、前出のコマンドの URL を `https://lambda-insights-extension-arm64.s3-ap-northeast-1.amazonaws.com/amazon_linux/lambda-insights-extension-arm64.1.0.229.0.rpm` に置き換えます。現時点では、Lambda Insights のバージョン 1.0.229.0 以降のみが利用可能です。詳細については、「Lambda Insights 拡張機能の利用可能なバージョン」を参照してください。

Linux サーバーで Lambda Insights エージェントパッケージの署名を確認するには

1. 次のコマンドを入力して、公開キーをダウンロードします。

   ```
   shell$ wget https://lambda-insights-extension-arm64.s3-ap-
   northeast-1.amazonaws.com/lambda-insights-extension.gpg
   ```

2. 次のコマンドを入力して、公開鍵をキーリングにインポートします。

   ```
   shell$ gpg --import lambda-insights-extension.gpg
   ```

   出力は以下のようになります。次のステップで必要になるため、key 値を書きとめておきます。この出力例では、キー値は 848ABDC8 です。

```
gpg: key 848ABDC8: public key "Amazon Lambda Insights Extension" imported
gpg: Total number processed: 1
gpg: imported: 1  (RSA: 1)
```

3. 次のコマンドを入力して、フィンガープリントを検証します。key-value の部分は、前の手順で記録したキーの値に置き換えます。

```
shell$  gpg --fingerprint key-value
```

このコマンドの出力では、フィンガープリント文字列は E0AF FA11 FFF3 5BD7 349E E222 479C 97A1 848A BDC8 のようになります。フィンガープリント文字列が一致しない場合は、エージェントのインストールを中止し、AWS にお問い合わせください。

4. フィンガープリントが確認できたら、それを使用して、Lambda Insights のエージェントパッケージを確認できます。次のコマンドを入力して、パッケージの署名ファイルをダウンロードします。

```
shell$  wget https://lambda-insights-extension-arm64.s3-ap-
northeast-1.amazonaws.com/amazon_linux/lambda-insights-extension-arm64.rpm.sig
```

5. 次のコマンドを入力して、署名を確認します。

```
shell$ gpg --verify lambda-insights-extension-arm64.rpm.sig lambda-insights-
extension-arm64.rpm
```

出力は次のようになります。

```
gpg: Signature made Thu 08 Apr 2021 06:41:00 PM UTC using RSA key ID 848ABDC8
gpg: Good signature from "Amazon Lambda Insights Extension"
gpg: WARNING: This key is not certified with a trusted signature!
gpg:          There is no indication that the signature belongs to the owner.
Primary key fingerprint: E0AF FA11 FFF3 5BD7 349E  E222 479C 97A1 848A BDC8
```

このときの出力には、信頼された署名に関する警告が表示される場合があります。キーは、自分や信頼する人が署名した場合にのみ信頼できます。つまり、署名が無効であるわけではなく、パブリックキーを確認していないことになります。

出力に BAD signature が含まれている場合は、ここまでの手順を正しく実行したかどうかを確認します。この BAD signature レスポンスが引き続き出力される場合は、AWS に連絡の上、ダウンロードしたファイルの使用は中止してください。

ARM64 の例

このセクションでは、コンテナイメージベースの Python Lambda 関数で Lambda Insights を有効にする例を示します。

Lambda コンテナイメージで Lambda Insights を有効にする例

1. 次のような Dockerfile を作成します。

```
FROM public.ecr.aws/lambda/python:3.8
// extra lines to install the agent here
RUN curl -O https://lambda-insights-extension-arm64.s3-ap-
northeast-1.amazonaws.com/amazon_linux/lambda-insights-extension-arm64.rpm && \
    rpm -U lambda-insights-extension-arm64.rpm && \
    rm -f lambda-insights-extension-arm64.rpm

COPY index.py ${LAMBDA_TASK_ROOT}
CMD [ "index.handler" ]
```

2. 次のような Python ファイルを、index.py という名前で作成します。

```
def handler(event, context):
  return {
    'message': 'Hello World!'
  }
```

3. Dockerfile と index.py を同じディレクトリに保存します。次に、保存先のディレクトリで次の手順を実行して Docker イメージをビルドし、それを Amazon ECR にアップロードします。

```
// create an ECR repository
aws ecr create-repository --repository-name test-repository
// build the docker image
docker build -t test-image .
// sign in to AWS
aws ecr get-login-password | docker login --username AWS --password-stdin
 "${ACCOUNT_ID}".dkr.ecr."${REGION}".amazonaws.com
// tag the image
```

```
docker tag test-image:latest "${ACCOUNT_ID}".dkr.ecr."${REGION}".amazonaws.com/
test-repository:latest
// push the image to ECR
docker push "${ACCOUNT_ID}".dkr.ecr."${REGION}".amazonaws.com/test-
repository:latest
```

4. 先ほど作成した Amazon ECR イメージを使用して、Lambda 関数を作成します。
5. CloudWatchLambdaInsightsExecutionRolePolicy IAM ポリシーを、関数の実行ロールに割り当てます。

Lambda Insights メトリクスの表示

呼び出された Lambda 関数に Lambda Insights 拡張機能をインストールしたら、CloudWatch コンソールを使用してメトリクスを確認できます。複数の関数の概要を表示することも、1 つの関数に集中することもできます。

Lambda Insights メトリクスのリストについては、「Lambda Insights によって収集されたメトリクス」を参照してください。

Lambda Insights メトリクスの複数の関数の概要を表示するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. 左側のナビゲーションペインの [Lambda Insights] で、[Multi-function (複数の関数)] を選択します。

   ページの上部には、Lambda Insights が有効になっているリージョン内のすべての Lambda 関数の集計されたメトリクスを含むグラフが表示されます。ページの下には、関数をリストした表があります。
3. 関数名でフィルターして表示される関数の数を減らすには、ページの上の方にあるボックスに関数名の一部を入力します。
4. このビューをウィジェットとしてダッシュボードに追加するには、[Add to dashboard (ダッシュボードに追加)] を選択します。

1 つの関数のメトリクスを表示するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. 左側のナビゲーションペインの [Lambda Insights] で、[Single-function (1 つの関数)] を選択します。

ページの上部に、選択した関数のメトリクスを含むグラフが表示されます。

3. X-Ray を有効にしている場合は、1 つのトレース ID を選択できます。これにより、その呼び出しの X-Ray トレースマップページを開き、そこからズームアウトして、分散トレースとその特定のトランザクションの処理に関係するその他のサービスを確認できます。X-Ray トレースマップの詳細については、「Using the X-Ray Trace Map」を参照してください。
4. CloudWatch Logs Insights を開いて特定のエラーにズームインするには、ページの下部にあるテーブルで [View logs (ログを表示する)] を選択します。
5. このビューをウィジェットとしてダッシュボードに追加するには、[Add to dashboard (ダッシュボードに追加)] を選択します。

Application Insights との統合

Amazon CloudWatch Application Insights は、アプリケーションのモニターリングに役立ち、アプリケーションリソースとテクノロジースタック全体の主要なメトリクス、ログ、アラームを識別して設定します。詳細については、「Amazon CloudWatch Application Insights」を参照してください。

Application Insights を有効にすると Lambda 関数から追加データを収集できます。まだ有効にしていない場合は、Lambda Insights ダッシュボードのパフォーマンスビューの下にある [Application Insights] タブで [Auto-configure Application Insights] (アプリケーションインサイトの自動構成) を選択して有効にできます。

Lambda 関数をモニターリングするように CloudWatch Application Insights を既に設定している場合は、[Application Insights] タブの Lambda Insights ダッシュボードの下に Application Insights ダッシュボードが表示されます。

Lambda Insights によって収集されたメトリクス

Lambda Insights は、インストールされている Lambda 関数からいくつかのメトリクスを収集します。これらのメトリクスの一部は、CloudWatch Metrics で時系列集計データとして使用できます。その他のメトリクスは、時系列データには集計されませんが、CloudWatch Logs Insights を使用して埋め込みメトリクスフォーマットのログエントリで確認できます。

次のメトリクスは、`LambdaInsights` 名前空間の CloudWatch Metrics で時系列集計データとして使用できます。

メトリクス名	ディメンション	説明
cpu_total_time	function_name function_name、バージョン	cpu_system_time と cpu_user_time の合計。 単位: ミリ秒
init_duration	function_name function_name、バージョン	Lambda 実行環境のライフサイクルの init フェーズで費やされた時間。 単位: ミリ秒
memory_utilization	function_name function_name、バージョン	最大メモリは、関数に割り当てられたメモリのパーセンテージとして測定されます。 単位: パーセント
rx_bytes	function_name function_name、バージョン	関数の受信バイト数。 単位: バイト
tmp_used		/tmp ディレクトリ内で使用される領域の量。 単位: バイト
tx_bytes	function_name function_name、バージョン	関数の送信バイト数。 単位: バイト

メトリクス名	ディメンション	説明
total_memory	function_name function_name、バージョン	Lambda 関数に割り当てられたメモリの量。これは関数のメモリサイズと同じです。 単位: メガバイト
total_network	function_name function_name、バージョン	rx_bytes と tx_bytes の合計。I/O タスクを実行しない関数の場合でも、Lambda ランタイムによって行われるネットワーク呼び出しのため、この値は通常 0 より大きくなります。 単位: バイト
used_memory_max	function_name function_name、バージョン	関数サンドボックスの測定されたメモリ。 単位: メガバイト

以下のメトリクスは、CloudWatch Logs Insights を使用して、埋め込みメトリクスフォーマットのログエントリにあります。CloudWatch Logs Insights の詳細については、「CloudWatch Logs Insights を使用したログデータの分析」を参照してください。

埋め込みメトリックフォーマットの詳細については、「ログ内へのメトリクスの埋め込み」を参照してください。

メトリクス名	説明
cpu_system_time	CPU がカーネルコードの実行に費やした時間。 単位: ミリ秒
cpu_total_time	cpu_system_time と cpu_user_time の合計。 単位: ミリ秒
cpu_user_time	CPU がユーザーコードの実行に費やした時間。 単位: ミリ秒
fd_max	使用可能なファイル記述子の最大数。 単位: 個
fd_use	使用中のファイル記述子の最大数。 単位: 個
memory_utilization	最大メモリは、関数に割り当てられたメモリのパーセンテージとして測定されます。 単位: パーセント
rx_bytes	関数の受信バイト数。 単位: バイト
tx_bytes	関数の送信バイト数。 単位: バイト
threads_max	関数プロセスで使用中のスレッドの数。関数の作成者は、ランタイムによって作成されたスレッドの初期数を制御しません。

メトリクス名	説明	
	単位: 個	
tmp_max	/tmp ディレクトリで使用可能な領域の量。 単位: バイト	
total_memory	Lambda 関数に割り当てられたメモリの量。これは関数のメモリサイズと同じです。 単位: メガバイト	
total_network	rx_bytes と tx_bytes の合計。I/O タスクを実行しない関数の場合でも、Lambda ランタイムによって行われるネットワーク呼び出しのため、この値は通常 0 より大きくなります。 単位: バイト	
used_memory_max	関数サンドボックスの測定されたメモリ。 単位: バイト	

トラブルシューティングと既知の問題

問題をトラブルシューティングする最初のステップは、Lambda Insights 拡張機能でデバッグログを有効にすることです。これを行うには、Lambda 関数に環境変数 LAMBDA_INSIGHTS_LOG_LEVEL=info を設定します。詳細については、AWS Lambda 環境変数の使用を参照してください。

拡張機能は、関数と同じロググループ (/aws/lambda/*function-name*) にログを放出します。これらのログを確認して、エラーがセットアップの問題に関連しているかどうかを確認します。

Lambda Insights からのメトリクスが表示されません

表示されるべき Lambda Insights メトリクスが表示されない場合は、次の可能性をチェックしてください。

- メトリクスが遅れている – 関数がまだ呼び出されていない場合、またはデータがまだフラッシュされていない場合は、メトリクスは CloudWatch に表示されません。詳細については、このセクションの後半の「既知の問題」をご参照ください。
- Lambda 関数が正しいアクセス許可を持っていることを確認する – CloudWatchLambdaInsightsExecutionRolePolicy IAM ポリシーが関数の実行ロールに割り当てられていることを確認します。
- Lambda ランタイムの確認 – Lambda Insights は特定の Lambda ランタイムのみをサポートします。サポートされているランタイムのリストについては、「Lambda Insights」を参照してください。

  例えば、Java 8 で Lambda Insights を使用するには、`java8` ランタイムではなく `java8.al2` ランタイムを使用する必要があります。
- ネットワークアクセスを確認する – この Lambda 関数は、インターネットにアクセスできない VPC プライベートサブネット上にあり、CloudWatch Logs のために VPC エンドポイントがに設定されていない可能性があります。この問題のデバッグに役立てるために、環境変数 `LAMBDA_INSIGHTS_LOG_LEVEL=info` を設定できます。

既知の問題

最長データ遅延は 20 分です。関数ハンドラーが完了すると、Lambda はサンドボックスをフリーズし、Lambda Insights 拡張機能もフリーズします。関数の実行中は、関数 TPS に基づくアダプティブバッチ戦略を使用してデータを出力します。ただし、関数が長期間呼び出されなくなり、バッファにイベントデータが残っている場合は、Lambda がアイドル状態のサンドボックスをシャットダウンするまでこのデータを遅延させることができます。Lambda がサンドボックスをシャットダウンすると、バッファされたデータはフラッシュされます。

テレメトリーイベントの例

Lambda Insights が有効になっている Lambda 関数の呼び出しごとに、`/aws/lambda-insights` ロググループに 1 つのログイベントが書き込まれます。各ログイベントには、埋め込みメトリックフォーマットにメトリクスが含まれます。埋め込みメトリックフォーマットの詳細については、「ログ内へのメトリクスの埋め込み」を参照してください。

これらのログイベントを分析するには、以下の方法を使用できます。

- CloudWatch コンソールの Lambda Insights セクション (「Lambda Insights メトリクスの表示」を参照)。

- CloudWatch Logs Insights を使用してイベントクエリをログに記録します。詳細については、Analyzing Log Data with CloudWatch Logs Insights を参照してください。
- `LambdaInsights` 名前空間で収集されるメトリクス。これは、CloudWatch メトリクスを使用してグラフ化します。

以下は、埋め込みメトリクスフォーマットの Lambda Insights ログイベントの例です。

```
{
    "_aws": {
        "Timestamp": 1605034324256,
        "CloudWatchMetrics": [
            {
                "Namespace": "LambdaInsights",
                "Dimensions": [
                    [ "function_name" ],
                    [ "function_name", "version" ]
                ],
                "Metrics": [
                    { "Name": "memory_utilization", "Unit": "Percent" },
                    { "Name": "total_memory", "Unit": "Megabytes" },
                    { "Name": "used_memory_max", "Unit": "Megabytes" },
                    { "Name": "cpu_total_time", "Unit": "Milliseconds" },
                    { "Name": "tx_bytes", "Unit": "Bytes" },
                    { "Name": "rx_bytes", "Unit": "Bytes" },
                    { "Name": "total_network", "Unit": "Bytes" },
                    { "Name": "init_duration", "Unit": "Milliseconds" }
                ]
            }
        ],
        "LambdaInsights": {
            "ShareTelemetry": true
        }
    },
    "event_type": "performance",
    "function_name": "cpu-intensive",
    "version": "Blue",
    "request_id": "12345678-8bcc-42f7-b1de-123456789012",
    "trace_id": "1-5faae118-12345678901234567890",
    "duration": 45191,
    "billed_duration": 45200,
    "billed_mb_ms": 11571200,
    "cold_start": true,
```

```
    "init_duration": 130,
    "tmp_free": 538329088,
    "tmp_max": 551346176,
    "threads_max": 11,
    "used_memory_max": 63,
    "total_memory": 256,
    "memory_utilization": 24,
    "cpu_user_time": 6640,
    "cpu_system_time": 50,
    "cpu_total_time": 6690,
    "fd_use": 416,
    "fd_max": 32642,
    "tx_bytes": 4434,
    "rx_bytes": 6911,
    "timeout": true,
    "shutdown_reason": "Timeout",
    "total_network": 11345,
    "agent_version": "1.0.72.0",
    "agent_memory_avg": 10,
    "agent_memory_max": 10
}
```


Contributor Insights を使用して高カーディナリティデータを分析する

Contributor Insights を使用して、ログデータを分析し、コントリビューターデータを表示する時系列を作成できます。トップ N コントリビューター、一意のコントリビューターの合計数、およびそれらの使用状況に関するメトリクスを確認できます。これを使用して、トップのトーカーを確認し、システムのパフォーマンスに影響を与えている人や物を判断できます。たとえば、不良ホストを検出したり、最も重いネットワークユーザーを特定したり、エラーを最も多く生成する URL を検索したりできます。

ルールは最初から構築するか、AWS Management Console で作成済みのサンプルルールを AWS コンソールで利用することができます。ルールは、`IpAddress` などのコントリビューターの定義に使用するログフィールドを定義します。また、ログデータをフィルタリングして、個別のコントリビューターの動作を見つけて分析できます。

CloudWatch はさらに、他の AWS サービスのメトリクスを分析するために使用できる組み込みルールを提供します。

すべてのルールが受信データをリアルタイムで分析します。

CloudWatch のクロスアカウントオブザーバビリティでモニターリングアカウントとして設定されたアカウントにサインインしている場合、そのモニターリングアカウントで、ソースアカウントとモニターリングアカウントのロググループを分析する Contributor Insights ルールを作成できます。複数のアカウントのロググループを分析する単一のルールを作成することもできます。詳細については、「CloudWatch のクロスアカウントオブザーバビリティ」を参照してください。

> **Note**
>
> Contributor Insights を使用すると、ルールに一致するログイベントが発生するたびに課金されます。詳細については、Amazon CloudWatch 料金表をご覧ください。

トピック

Contributor Insights ルールの作成

ルールを作成してログデータを分析できます。JSON または Common Log Format (CLF) のログはすべて評価できます。これには、これらの形式のいずれかに従うカスタムログと、Amazon VPC フローログ、Amazon Route 53 DNS クエリログ、Amazon ECS コンテナログ、および AWS CloudTrail、Amazon SageMaker、Amazon RDS、AWS AppSync、API Gateway からのログなど、AWS のサービスからのログが含まれます。

ルールでは、フィールド名または値を指定する場合、すべてのマッチングで大文字と小文字が区別されます。

組み込みのサンプルルールを使用してルールを作成したり、独自のルールを最初から作成したりできます。Contributor Insights には、次の種類のログのサンプルルールが含まれています。

- Amazon API Gateway ログ
- Amazon Route 53 パブリック DNS クエリログ

- Amazon Route 53 Resolver クエリログ
- CloudWatch Container Insights ログ
- VPC フローログ

CloudWatch のクロスアカウントオブザーバビリティでモニターリングアカウントとして設定されているアカウントにサインインしている場合は、モニターリングアカウントのロググループに対するルールを作成できるだけでなく、このモニターリングアカウントにリンクされているソースアカウントのロググループに対しても Contributor Insights ルールを作成できます。単一のルールを設定して、異なるアカウントのロググループをモニターリングすることもできます。詳細については、「CloudWatch のクロスアカウントオブザーバビリティ」を参照してください。

> **Important**
>
> ユーザーに `cloudwatch:PutInsightRule` アクセス許可を付与すると、デフォルトでは、そのユーザーは CloudWatch Logs 内の任意のロググループを評価するルールを作成できます。特定のロググループを含める、および除外するユーザーのアクセス許可を制限する IAM ポリシー条件を追加できます。詳細については、「」を参照してくださいContributor Insights のユーザーのロググループへのアクセスを制限するための条件キーの使用

組み込みのサンプルルールを使用してルールを作成するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで、[Insights] (インサイト)、[Contributor Insights] の順に選択します。
3. ルールの作成を選択します。

4. [Select log group(s)] (ロググループを選択) で、ルールでモニターリングするロググループを選択します。ロググループは 20 個まで選択できます。CloudWatch のクロスアカウントオブザーバビリティのために設定されたモニターリングアカウントにサインインしている場合は、ソースアカウントのロググループを選択できるだけでなく、単一のルールを設定して異なるアカウントのロググループを分析することもできます。

   - (オプション) 名前が特定の文字列で始まるすべてのロググループを選択するには、[Select by prefix match] (プレフィックスの一致による選択) ドロップダウンを選択し、プレフィックスを入力します。これがモニターリングアカウントの場合は、必要に応じて検索対象のアカウントを選択できます。そうでない場合は、すべてのアカウントが選択されます。

> **Note**
>
> ルールと一致するログイベントごとに料金が発生します。[Select by prefix match] (プレフィックスの一致による選択) ドロップダウンを選択する場合、プレフィクスが一致する可能性があるロググループの数に注意してください。意図したより多くのロググループを検索すると、想定外の料金が発生することがあります。詳細については、「Amazon CloudWatch 料金表」をご覧ください。

5. [Rule type] (ルールタイプ) で、[Sample rule] (サンプルルール) を選択します。次に、[Select sample rule] (サンプルルールの選択) を選択し、ルールを選択します。
6. サンプルルールでは、ログの形式、寄与度、フィルター、および集計の各フィールドが入力済みです。これらの値は、必要に応じて調整できます。
7. [Next] を選択します。
8. [Rule name] に名前を入力します。有効な文字は、A ~ Z、a ~ z、0 ~ 9、(ハイフン)、(アンダースコア)、(ピリオド) です。
9. ルールを無効状態で作成するか有効状態で作成するかを選択します。ルールを有効にすると、即座にデータの分析が開始されます。有効にしたルールを実行すると、コストが発生します。詳細については、Amazon CloudWatch 料金表をご覧ください。

   Contributor Insights は、ルールが作成されたあとに新しいログイベントのみを分析します。ルールは、以前に CloudWatch Logs によって処理されたログイベントは処理できません。
10. (オプション) [Tags] (タグ) で、このルールのタグとする 1 つまたは複数のキーと値のペアを追加します。タグを使用すると、AWS リソースを識別して整理したり、AWS コストを追跡したりしやすくなります。詳細については、「Amazon CloudWatch リソースにタグを付ける」を参照してください。
11. [Create] (作成) を選択します。

ルールを最初から作成するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで、[Contributor Insights] を選択します。
3. ルールの作成を選択します。

4. [Select log group(s)] (ロググループを選択) で、ルールでモニターリングするロググループを選択します。ロググループは 20 個まで選択できます。CloudWatch のクロスアカウントオブザーバビリティのために設定されたモニターリングアカウントにサインインしている場合は、ソースアカウントのロググループを選択できるだけでなく、単一のルールを設定して異なるアカウントのロググループを分析することもできます。

   - (オプション) 名前が特定の文字列で始まるすべてのロググループを選択するには、[Select by prefix match] (プレフィックスの一致による選択) ドロップダウンを選択し、プレフィックスを入力します。

   > **Note**
   >
   > ルールと一致するログイベントごとに料金が発生します。[Select by prefix match] (プレフィックスの一致による選択) ドロップダウンを選択する場合、プレフィクスが一致する可能性があるロググループの数に注意してください。意図したより多くのロググループを検索すると、想定外の料金が発生することがあります。詳細については、「Amazon CloudWatch 料金表」をご覧ください。

5. [Rule type] (ルールタイプ) で [Custom rule] (カスタムルール) を選択します。
6. [Log Format] で、[JSON] または [CLF] を選択します。
7. ルールの作成を完了するには、ウィザードを使用するか、[Syntax] タブを選択して手動でルール構文を指定します。

   ウィザードの使用を続けるには、次の操作を実行します。

   a. [Contribution]、[Key] に、レポート対象のコントリビュータータイプを入力します。このコントリビュータータイプのトップ N の値がレポートに表示されます。

      値を持つログフィールドが有効なエントリです。例には、**`requestId`**、**`sourceIPaddress`**、**`containerID`** が含まれます。

      特定のロググループに属するログのログフィールド名を検索する方法については、「ログフィールドの検索」を参照してください。

      1 KB 以上のキーは切り捨てられ、1 KB になります。

   b. (オプション) [Add new key] (新しいキーを追加) を選択して、キーを追加します。ルールには、最大 4 つのキーを含められます。複数のキーを入力すると、レポートのコントリビューターは、これらのキーの一意な値の組み合わせで定義されます。たとえば、3 つ

のキーを指定した場合、3 つのキーの一意の値の組み合わせそれぞれが、一意のコントリビューターとしてカウントされます。

c. (オプション) 結果の範囲を狭めるフィルターを追加する場合は、[Add filter] (フィルターを追加) を選択します。[Match] (一致) で、フィルタリングするログフィールドの名前を入力します。[Condition] (条件) で、比較演算子を選択し、フィルタリングする値を入力します。

ルールには、最大 4 つのフィルターを追加できます。複数のフィルターは AND ロジックで結合されるため、すべてのフィルターに一致するログイベントのみが評価されます。

> **Note**
>
> `In`、`NotIn`、`StartsWith` などの比較演算子に続く配列には、最大 10 個の文字列値を含めることができます。Contributor Insights ルールの構文の詳細については、「Contributor Insights のルール構文」を参照してください。

d. [Aggregate on] (集計) で、[Count] (カウント) または [Sum] (合計) を選択します。[Count] (カウント) を選択すると、コントリビューターランキングが出現回数に基づきます。[Sum] (合計) を選択すると、ランキングは [Contribution] (コントリビューション)、[Value] (値) に指定したフィールドの値を集約した合計に基づきます。

8. ウィザードを使用する代わりにルールを JSON オブジェクトとして入力するには、次の手順を実行します。

a. [Syntax] タブを選択します。

b. [Rule body] に、ルールの JSON オブジェクトを入力します。ルール構文の詳細については、「Contributor Insights のルール構文」を参照してください。

9. [Next] を選択します。

10. [Rule name] に名前を入力します。有効な文字は、A ~ Z、a ~ z、0 ~ 9、- (ハイフン)、_ (アンダースコア)、. (ピリオド) です。

11. ルールを無効状態で作成するか有効状態で作成するかを選択します。ルールを有効にすると、即座にデータの分析が開始されます。有効にしたルールを実行すると、コストが発生します。詳細については、Amazon CloudWatch 料金表をご覧ください。

Contributor Insights は、ルールが作成されたあとに新しいログイベントのみを分析します。ルールは、以前に CloudWatch Logs によって処理されたログイベントは処理できません。

12. (オプション) [Tags] (タグ) で、このルールのタグとする 1 つまたは複数のキーと値のペアを追加します。タグを使用すると、AWS リソースを識別して整理したり、AWS コストを追跡した

りしやすくなります。詳細については、「Amazon CloudWatch リソースにタグを付ける」を参照してください。

13. [Next] を選択します。
14. 入力した設定を確認し、[Create rule] (ルールの作成) を選択します。

作成したルールは、無効化、有効化、または削除できます。

Contributor Insights のルールを有効化、無効化、または削除するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで、[Contributor Insights] を選択します。
3. ルールのリストで、1 つのルールの横にあるチェックボックスをオンにします。

   組み込みルールは AWS サービスによって作成され、編集、無効化、または削除できません。
4. [アクション] を選択し、必要なオプションを選択します。

ログフィールドの検索

ルールを作成するときは、ロググループのログエントリのフィールド名を知っている必要があります。

ロググループでログフィールドを検索するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインの [ログ] で、[Insights] を選択します。
3. クエリエディタの上で、クエリするロググループを 1 つ以上選択します。

   ロググループを選択すると、CloudWatch Logs Insights はロググループのデータのフィールドを自動的に検出して、[Discovered fields] の右側のペインに表示します。

Contributor Insights のルール構文

このセクションでは、Contributor Insights ルールの構文について説明します。この構文は、JSON ブロックを入力してルールを作成するときのみ使用します。ウィザードを使ってルールを作成する場合は、構文を知る必要はありません。ウィザードを使ってルールを作成する詳しい方法については、「Contributor Insights ルールの作成」を参照してください。

ログイベントフィールド名および値に対するルールのすべてのマッチングでは、大文字と小文字が区別されます。

次の例は JSON ログの構文を説明します。

```
{
    "Schema": {
        "Name": "CloudWatchLogRule",
        "Version": 1
    },
    "LogGroupNames": [
        "API-Gateway-Access-Logs*",
        "Log-group-name2"
    ],
    "LogFormat": "JSON",
    "Contribution": {
        "Keys": [
            "$.ip"
        ],
        "ValueOf": "$.requestBytes",
        "Filters": [
            {
                "Match": "$.httpMethod",
                "In": [
                    "PUT"
                ]
            }
        ]
    },
    "AggregateOn": "Sum"
}
```

Contributor Insights ルールのフィールド

Schema

CloudWatch Logs データを分析するルールの Schema の値は、常に {"Name": "CloudWatchLogRule", "Version": 1} である必要があります。

LogGroupNames

文字列の配列です。配列の各要素で、必要に応じて文字列の最後に * を使用して、そのプレフィックスで始まる名前を持つすべてのロググループを含めることができます。

ロググループ名でワイルドカードを使用するときはご注意ください。ルールと一致するログイベントごとに料金が発生します。誤って意図したより多くのロググループを検索すると、予期しない料金が発生することがあります。詳細については、Amazon CloudWatch 料金表をご覧ください。

LogGroupARNs

CloudWatch のクロスアカウントオブザーバビリティモニターリングアカウントでこのルールを作成する場合、LogGroupARNs を使用して、モニターリングアカウントにリンクされているソースアカウントのロググループを指定したり、モニターリングアカウント自体のログループを指定したりできます。ルール内で LogGroupNames または LogGroupARNs を指定する必要があります。ただし、両方を指定することはできません。

LogGroupARNs は文字列の配列です。配列の各要素で、特定の状況においては、必要に応じて * をワイルドカードとして使用できます。例えば、arn:aws:logs:us-west-1:*:log-group/MyLogGroupName2 を指定すると、米国西部 (北カリフォルニア) リージョン内のすべてのソースアカウントとモニターリングアカウントで MyLogGroupName2 という名前のロググループを指定できます。また、arn:aws:logs:us-west-1:111122223333:log-group/GroupNamePrefix* を指定すると、名前が GroupNamePrefix で始まる米国西部 (北カリフォルニア) 内の 111122223333 のロググループすべてを指定することもできます。

AWS アカウント ID の一部を、ワイルドカードが付いたプレフィックスとして指定することはできません。

ロググループ ARN でワイルドカードを使用するときはご注意ください。ルールと一致するログイベントごとに料金が発生します。誤って意図したより多くのロググループを検索すると、予期しない料金が発生することがあります。詳細については、Amazon CloudWatch 料金表をご覧ください。

LogFormat

有効な値は、JSON および CLF です。

寄与度

このオブジェクトには、最大で 4 つのメンバーがある Keys 配列、オプションで 1 つの ValueOf、およびオプションで最大 4 つの Filters がある配列が含まれます。

キー

最大 4 つのログフィールドの配列であり、コントリビューターを分類するためのディメンションとして使用されます。複数のキーを入力すると、これらのキーの一意な値の組み合わせごとに一

意のコントリビューターとしてカウントされます。フィールドは、JSON プロパティフォーマット表記を使用して指定する必要があります。

ValueOf

(オプション) Sum を AggregateOn の値として指定する場合のみこれを指定します。ValueOf は数値を持つログフィールドを指定します。このタイプのルールでは、コントリビューターはログエントリでの出現回数ではなく、このフィールドの値の合計によってランク付けされます。たとえば、コントリビューターを特定期間にわたる合計 BytesSent でソートする場合は、ValueOf を BytesSent に設定し、Sum を AggregateOn に指定します。

フィルター

(オプション) 最大 4 つのフィルターの配列を指定し、レポートに含まれるログイベントを絞り込みます。複数のフィルターを指定した場合、Contributor Insights は論理 AND 演算子で評価します。これを使用して、検索から無関係なログイベントを除外したり、1 つのコントリビューターを選択して動作を分析したりできます。

配列内の各メンバーは、Match フィールドと、使用する一致する演算子のタイプを示すフィールドを含める必要があります。

Match フィールドは、フィルターで評価するログフィールドを指定します。ログフィールドは JSON プロパティフォーマット表記を使用して指定されます。

一致する演算子フィールド
は、In、NotIn、StartsWith、GreaterThan、LessThan、EqualTo、NotEqualTo、または IsPresent のいずれかである必要があります。演算子フィールドがIn、NotIn、または StartsWith である場合、チェックする文字列値の配列が続きます。Contributor Insights は文字列値の配列を OR 演算子で評価します。配列には最大で 10 個の文字列値を含めることができます。

演算子フィールドが GreaterThan、LessThan、EqualTo、または NotEqualTo である場合、比較する 1 つの数値が続きます。

演算子フィールドが IsPresent である場合、true または false が続きます。この演算子は、指定したログフィールドがログイベント内にあるかどうかに基づいて、ログイベントを照合します。isPresent は JSON プロパティの葉ノードの値に対してのみ機能します。たとえば、c-count との一致を検索するフィルターは、details.c-count.c1 を値とするログイベントを評価しません。

次の 4 つのフィルターの例を参照してください。

```
{"Match": "$.httpMethod", "In": [ "PUT", ] }
{"Match": "$.StatusCode", "EqualTo": 200 }
{"Match": "$.BytesReceived", "GreaterThan": 10000}
{"Match": "$.eventSource", "StartsWith": [ "ec2", "ecs" ] }
```

AggregateOn

有効な値は、Count および Sum です。出現回数に基づいてレポートを集計するか、ValueOf フィールドで指定されたフィールドの値の合計に基づいてレポートを集計するかを指定します。

JSON プロパティフォーマット表記

Keys、ValueOf、および Match フィールドはドット表記の JSON プロパティフォーマットに従い、$ は JSON オブジェクトのルートを表します。このあとにはピリオドと、サブプロパティの名前を含む英数字の文字列が続きます。複数のプロパティレベルがサポートされています。

文字列の最初の文字は A-Z または a-z だけです。文字列の次の文字は、A ~ Z、a ~ z、0 ~ 9 のいずれでもかまいません。

次のリストは、JSON プロパティフォーマットの有効な例を示しています。

```
$.userAgent
$.endpoints[0]
$.users[1].name
$.requestParameters.instanceId
```

CLF ログのルールの追加フィールド

Common Log Format (CLF) ログイベントには、JSON にはあるようなフィールドの名前はありません。Contributor Insights ルールで使用するフィールドを提供するために、CLF ログイベントを 1 で始まるインデックスを持つ配列として扱うことができます。最初のフィールドを **"1"** として指定し、2 番目のフィールドを **"2"** として指定できます。以下同様です。

CLF ログのルールを読みやすくするには、Fields を使用できます。これにより、CLF フィールドの位置の命名エイリアスを提供することができます。たとえば、位置「4」が IP アドレスであることを指定できます。指定すると、IpAddress をルールで Keys、ValueOf、および Filters のプロパティとして使用できます。

次の例は、Fields フィールドを使用する CLF ログのルールを示しています。

```
{
```

```
    "Schema": {
        "Name": "CloudWatchLogRule",
        "Version": 1
    },
    "LogGroupNames": [
        "API-Gateway-Access-Logs*"
    ],
    "LogFormat": "CLF",
    "Fields": {
        "4": "IpAddress",
        "7": "StatusCode"
    },
    "Contribution": {
        "Keys": [
            "IpAddress"
        ],
        "Filters": [
            {
                "Match": "StatusCode",
                "EqualTo": 200
            }
        ]
    },
    "AggregateOn": "Count"
}
```

Contributor Insights ルールの例

このセクションには、Contributor Insights ルールのユースケースを示す例が含まれています。

VPC フローログ: 送信元 IP アドレスおよび送信先 IP アドレスごとのバイト転送

```
{
    "Schema": {
        "Name": "CloudWatchLogRule",
        "Version": 1
    },
    "LogGroupNames": [
        "/aws/containerinsights/sample-cluster-name/flowlogs"
    ],
    "LogFormat": "CLF",
    "Fields": {
        "4": "srcaddr",
```

```
        "5": "dstaddr",
        "10": "bytes"
    },
    "Contribution": {
        "Keys": [
            "srcaddr",
            "dstaddr"
        ],
        "ValueOf": "bytes",
        "Filters": []
    },
    "AggregateOn": "Sum"
}
```

VPC フローログ: HTTPS リクエストの最高数

```
{
    "Schema": {
        "Name": "CloudWatchLogRule",
        "Version": 1
    },
    "LogGroupNames": [
        "/aws/containerinsights/sample-cluster-name/flowlogs"
    ],
    "LogFormat": "CLF",
    "Fields": {
        "5": "destination address",
        "7": "destination port",
        "9": "packet count"
    },
    "Contribution": {
        "Keys": [
            "destination address"
        ],
        "ValueOf": "packet count",
        "Filters": [
            {
                "Match": "destination port",
                "EqualTo": 443
            }
        ]
    },
    "AggregateOn": "Sum"
```

```
}
```

VPC フローログ: 拒否された TCP 接続

```
{
    "Schema": {
        "Name": "CloudWatchLogRule",
        "Version": 1
    },
    "LogGroupNames": [
        "/aws/containerinsights/sample-cluster-name/flowlogs"
    ],
    "LogFormat": "CLF",
    "Fields": {
        "3": "interfaceID",
        "4": "sourceAddress",
        "8": "protocol",
        "13": "action"
    },
    "Contribution": {
        "Keys": [
            "interfaceID",
            "sourceAddress"
        ],
        "Filters": [
            {
                "Match": "protocol",
                "EqualTo": 6
            },
            {
                "Match": "action",
                "In": [
                    "REJECT"
                ]
            }
        ]
    },
    "AggregateOn": "Sum"
}
```

送信元アドレスによる Route 53 NXDomain 応答

```
{
```

```
    "Schema": {
        "Name": "CloudWatchLogRule",
        "Version": 1
    },
    "AggregateOn": "Count",
    "Contribution": {
        "Filters": [
            {
                "Match": "$.rcode",
                "StartsWith": [
                    "NXDOMAIN"
                ]
            }
        ],
        "Keys": [
            "$.srcaddr"
        ]
    },
    "LogFormat": "JSON",
    "LogGroupNames": [
        "<loggroupname>"
    ]
}
```

ドメイン名による Route 53 Resolver クエリ

```
{
    "Schema": {
        "Name": "CloudWatchLogRule",
        "Version": 1
    },
    "AggregateOn": "Count",
    "Contribution": {
        "Filters": [],
        "Keys": [
            "$.query_name"
        ]
    },
    "LogFormat": "JSON",
    "LogGroupNames": [
        "<loggroupname>"
    ]
}
```

クエリタイプと送信元アドレスによる Route 53 Resolver クエリ

```
{
    "Schema": {
        "Name": "CloudWatchLogRule",
        "Version": 1
    },
    "AggregateOn": "Count",
    "Contribution": {
        "Filters": [],
        "Keys": [
            "$.query_type",
            "$.srcaddr"
        ]
    },
    "LogFormat": "JSON",
    "LogGroupNames": [
        "<loggroupname>"
    ]
}
```

Contributor Insights レポートの表示

レポートデータのグラフや、ルールによって見つかったコントリビューターのランク付けリストを表示するには、次の手順を実行します。

ルールレポートを表示するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで、[Contributor Insights] を選択します。
3. ルールのリストで、ルールの名前を選択します。

   グラフには、過去 3 時間のルールの結果が表示されます。グラフの下にある表には、トップ 10 コントリビューターを表示します。
4. 表に表示されるコントリビューターの数を変更するには、表の上部にある [Top 10 contributors] を選択します。
5. 表に単一のコントリビューターの結果のみを表示するようフィルタリングするには、表の凡例でそのコントリビューターを選択します。再びすべてのコントリビューターを表示するには、凡例で同じコントリビューターを再び選択します。

6. レポートに表示される時間範囲を変更するには、表の上部で [15m] (15 分)、[30m] (30 分)、[1h] (1 時間)、[2h] (2 時間)、[3h] (3 時間)、または [custom] (カスタム) を選択します。

   レポートの最大時間範囲は 24 時間ですが、最大 15 日前に発生した 24 時間の時間窓を選択できます。過去の時間窓を選択するには、[custom]、[absolute] を選択し、時間窓を指定します。
7. コントリビューターの集計とランク付けに使用する期間を変更するには、表の上部で [period] を選択します。より長い期間を表示すると、大抵の場合スパイクが少ない滑らかなレポートが表示されます。短い期間を選択すると、スパイクが表示される可能性が高くなります。
8. このグラフを CloudWatch ダッシュボードに追加するには、[ダッシュボードに追加] を選択します。
9. CloudWatch Logs Insights クエリウィンドウを開くには、このレポートのロググループがクエリボックスにロード済みの状態で、[ログの表示] を選択します。
10. レポートデータをクリップボードまたは CSV ファイルにエクスポートするには、[Export] を選択します。

ルールによって生成されたメトリクスのグラフ化

Contributor Insights にはメトリクス数学関数、INSIGHT_RULE_METRIC があります。この関数を使用して、Contributor Insights レポートのデータを、CloudWatch コンソールの [メトリクス] タブにあるグラフに追加できます。また、この数学関数に基づいてアラームを設定することもできます。メトリクス数学関数の詳細については、「Metric Math を使用する」を参照してください。

このメトリクス数学関数を使用するには、cloudwatch:GetMetricData アクセス許可と cloudwatch:GetInsightRuleReport アクセス許可の両方を持つアカウントにサインインしている必要があります。

構文は INSIGHT_RULE_METRIC(*ruleName*, *metricName*) です。[*ruleName*] は Contributor Insights ルールの名前です。[*metricName*] は次のリストにある値の 1 つです。[*metricName*] の値は、math 関数が返すデータのタイプを決定します。

- UniqueContributors – 各データポイントの一意のコントリビューターの数です。
- MaxContributorValue – 各データポイントのトップコントリビューターの値です。コントリビューターの ID は、グラフのデータポイントごとに変わる場合があります。

  このルールを Count によって集計する場合、各データポイントのトップコントリビューターはその期間中に最も多く出現したコントリビューターです。ルールを Sum によって集計する場合、

トップコントリビューターはその期間中にルールの Value によって指定されたログフィールドで最大合計を持つコントリビューターです。

- SampleCount – ルールと一致するデータポイントの数です。
- Sum – そのデータポイントによって表される期間中のすべてのコントリビューターの値の合計です。
- Minimum – そのデータポイントによって表される期間中の単一の観測値の最小値です。
- Maximum – そのデータポイントによって表される期間中の単一の観測値の最大値です。
- Average – そのデータポイントによって表される期間中のすべてのコントリビューターの平均値です。

Contributor Insights メトリクスデータでのアラームの設定

この関数 INSIGHT_RULE_METRIC を使用すると、Contributor Insights が生成するメトリクスにアラームを設定できます。例えば、拒否された Transmission Control Protocol (TCP) 接続の割合に基づくアラームを作成できます。このタイプのアラームの使用を開始するために、次の 2 つの例に示すようなルールを作成できます。

ルールの例:「RejectedConnectionsRule」

```
{
    "Schema": {
        "Name": "CloudWatchLogRule",
        "Version": 1
    },
    "LogGroupNames": [
        "/aws/containerinsights/sample-cluster-name/flowlogs"
    ],
    "LogFormat": "CLF",
    "Fields": {
        "3": "interfaceID",
        "4": "sourceAddress",
        "8": "protocol",
        "13": "action"
    },
    "Contribution": {
        "Keys": [
            "interfaceID",
            "sourceAddress"
        ],
```

```
        "Filters": [
            {
                "Match": "protocol",
                "EqualTo": 6
            },
            {
                "Match": "action",
                "In": [
                    "REJECT"
                ]
            }
        ]
    },
    "AggregateOn": "Sum"
}
```

ルールの例:「TotalConnectionsRule」

```
{
    "Schema": {
        "Name": "CloudWatchLogRule",
        "Version": 1
    },
    "LogGroupNames": [
        "/aws/containerinsights/sample-cluster-name/flowlogs"
    ],
    "LogFormat": "CLF",
    "Fields": {
        "3": "interfaceID",
        "4": "sourceAddress",
        "8": "protocol",
        "13": "action"
    },
    "Contribution": {
        "Keys": [
            "interfaceID",
            "sourceAddress"
        ],
        "Filters": [
            {
                "Match": "protocol",
                "EqualTo": 6
            }
```

```
    "AggregateOn": "Sum"
}
```

ルールを作成したら、CloudWatch コンソールで [Metrics] (メトリクス) タブを選択できます。このタブでは、次のメトリクス数式の例を使用して、Contributor Insights がレポートするデータをグラフ化できます。

例: メトリクス数式

```
e1 INSIGHT_RULE_METRIC("RejectedConnectionsRule", "Sum")
e2 INSIGHT_RULE_METRIC("TotalConnectionsRule", "Sum")
e3 (e1/e2)*100
```

この例では、メトリクス数式 e3 は、拒否されたすべての TCP 接続を返します。TCP 接続の 20% が拒否されたときに通知を受け取りたい場合は、しきい値を 100 から 20 に変更して式を変更できます。

> **Note**
>
> [Metrics] (メトリクス) セクションから、モニターリングしているメトリクスにアラームを設定できます。[Graphed metrics] (グラフ化されたメトリクス) タブで、[Actions] (アクション) 列の下にある [Create alarm] (アラームを作成) アイコンを選択できます。[Create alarm] (アラームを作成) アイコンはベルのような見た目です。

表メトリクスと Metric Math 関数の使用に関する詳細については、「CloudWatch グラフに数式を追加する」のセクションを参照してください。

Contributor Insights 組み込みルールの使用

Contributor Insights 組み込みルールを使用して、他の AWS サービスからメトリクスを分析できます。次のサービスでは組み込みルールをサポートします。

- Amazon DynamoDB 開発者ガイドの「Amazon DynamoDB の Contributor Insights」。
- AWS PrivateLink ガイドの「組み込み Contributor Insights ルールを使用する」。

Amazon CloudWatch Application Insights

Amazon CloudWatch Application Insights は、アプリケーションと、その基盤となる AWS のリソースに対するオブザーバビリティを向上させます。アプリケーションのリソースを監視する最適な条件を設定し、データを継続的に分析してアプリケーションの問題の徴候を検出できます。Application Insights は、SageMaker などの AWS テクノロジーを基盤とし、アプリケーションをモニターリングして検出した潜在的な問題を自動ダッシュボードに表示します。これにより、アプリケーションやインフラストラクチャで進行中の問題をすばやく特定できます。Application Insights が提供するアプリケーションの正常性に関する強化された可視性は、アプリケーションの問題のトラブルシューティングに要する平均修復時間 (MTTR) を低減するのに役立ちます。

Amazon CloudWatch Application Insights にアプリケーションを追加すると、アプリケーションのリソースがスキャンされて、アプリケーションコンポーネントに対して推奨されるメトリクスとログが CloudWatch に設定されます。アプリケーションコンポーネントの例には、SQS Server バックエンドデータベースと Microsoft IIS/Web 層があります。Application Insights は、履歴データを使用してメトリクスのパターンを分析し、異常を検出します。また、アプリケーション、オペレーティングシステム、インフラストラクチャログからエラーや例外を継続的に検出します。これらの監視結果は、分類アルゴリズムと組み込みルールの組み合わせを使用して相互に関連付けられます。その後、自動作成されるダッシュボードに表示されるモニターリング結果と問題の重大度に関する情報に基づいて、対処するアクションの優先順位を決定できます。.NET および SQL アプリケーションスタックの一般的な問題 (アプリケーションのレイテンシー、SQL Server のバックアップの失敗、メモリリーク、大きな HTTP リクエスト、I/O オペレーションのキャンセルなど) については、根本原因を示唆する追加のインサイトと解決の手順が示されます。組み込まれている AWS SSM OpsCenter との統合により、関連する Systems Manager Automation ドキュメントを実行して問題を解決できます。

セクション

- チュートリアル: SAP NetWeaver のモニターリングを設定する
- Amazon CloudWatch Application Insights での検出された問題の表示とトラブルシューティング
- Amazon CloudWatch Application Insights でサポートされるログとメトリクス

Amazon CloudWatch Application Insights とは

CloudWatch Application Insights は、他のアプリケーションリソースとともに Amazon EC2 インスタンスを使用するアプリケーションをモニターリングするのに役立ちます。アプリケーションのリソース全体およびテクノロジースタック (Microsoft SQL Server データベース、ウェブ (IIS) サーバー、アプリケーションサーバー、OS、ロードバランサー、キューなど) で主要なメトリクス、ログ、およびアラームを特定して設定します。メトリクスとログを継続的にモニターリングし、異常やエラーを検出して相互に関連付けます。エラーや異常が検出されると、Application Insights は CloudWatch Events を生成します。これを使用して、通知を設定したり、アクションを実行したりできます。トラブルシューティングに役立てるために、検出した問題の自動ダッシュボードを作成します。このダッシュボードには、相互に関連付けられた異常とログエラー、さらに根本原因を示唆する追加のインサイトが示されます。自動ダッシュボードを使用すると、是正措置を簡単に実行してアプリケーションを正常な状態に保ち、アプリケーションのエンドユーザーへの影響を防止できます。OpsItems も作成されるため、AWS SSM OpsCenter を使用して問題を解決できます。

ミラーリングされた書き込みトランザクション/秒、リカバリキューの長さ、トランザクション遅延などの重要なカウンター、CloudWatch での Windows イベントログを設定できます。SQL HA ワークロードでフェイルオーバーイベントや問題 (ターゲットデータベースのクエリに対するアクセスの制限など) が発生した場合、CloudWatch Application Insights は自動化されたインサイトを提供します。

CloudWatch Application Insights は AWS Launch Wizard との統合により、SQL Server HA ワークロードを AWS にデプロイするためのモニターリングを 1 回のクリック操作で設定できます。Launch Wizard コンソールの Application Insights でモニターリングおよびインサイトを設定するオプションを選択すると、CloudWatch Application Insights は関連するメトリクス、ログ、アラームを CloudWatch で自動的に設定し、新しくデプロイされたワークロードのモニターリングを開始します。自動化されたインサイトと検出された問題は、SQL Server HA ワークロードのヘルスとともに CloudWatch コンソールで確認できます。

内容

- 機能
- 概念

- 料金
- 関連する のサービス
- サポートされるアプリケーションコンポーネント
- サポートされるテクノロジースタック

機能

Application Insights には、次の機能があります。

アプリケーションリソースのモニターリングの自動設定

CloudWatch Application Insights を使用すると、アプリケーションのモニターリングの設定にかかる時間が短縮されます。CloudWatch Application Insights for.NET and SQL Server は、アプリケーションリソースをスキャンして推奨されるメトリクスとログのリスト (カスタマイズ可能) を提供し、CloudWatch でこのメトリクスやログを設定することで、Amazon EC2 および Elastic Load Balancing (ELB) などのアプリケーションリソースに必要な可視性を提供します。また、モニターリングしたメトリクスに対して動的なアラームを設定します。アラームは、過去 2 週間に検出された異常に基づいて自動的に更新されます。

問題の検出と通知

CloudWatch Application Insights は、メトリクスの異常やログエラーなど、アプリケーションの潜在的な問題の兆候を検出します。これらの監視結果を相互に関連付けることで、アプリケーションの潜在的な問題を表面化させます。次に、CloudWatch Events を生成します。これらのイベントは、通知を送信するか、アクションを実行するように設定できます。これにより、メトリクスやログエラーごとに個別のアラームを作成する必要がなくなります。

トラブルシューティング

CloudWatch Application Insights は、検出された問題用の CloudWatch 自動ダッシュボードを作成します。このダッシュボードには、問題に関する詳細として、トラブルシューティングに役立つ関連メトリクスの異常やログエラーなどが表示されます。また、異常やエラーの根本原因を示唆する追加のインサイトも示されます。

概念

以下の概念は、Application Insights によるアプリケーションのモニターリング方法を理解するうえで重要です。

コンポーネント

アプリケーションを構成する類似リソースの自動グループ化、スタンドアロン、またはカスタムグループ化。モニターリングを向上させるために、類似したリソースをカスタムコンポーネントとしてグループ化することをお勧めします。

監視結果

アプリケーションやアプリケーションリソースで検出された個別のイベント (メトリクスの異常、ログエラー、または例外)。

問題

問題は、関連する監視結果の相互関連付け、分類、およびグループ化によって検出されます。

CloudWatch Application Insights に関する他の主要な概念の定義については、「Amazon CloudWatch の概念」を参照してください。

料金

CloudWatch Application Insights は、検出した問題を通知するために CloudWatch メトリクス、CloudWatch Logs、および CloudWatch Events を使用して、選択したアプリケーションリソースに推奨されるメトリクスとログを設定します。これらの機能は、CloudWatch の料金に基づいて AWS アカウントに請求されます。検出した問題については、Application Insights によって SSM OpsItem も作成され、ユーザーはその問題に関する通知を受け取ることができます。さらに、Application Insights は、SSM パラメータストアのパラメータを作成して、インスタンスで CloudWatch エージェントを設定します。Amazon EC2 Systems Manager の機能は SSM の料金に従って課金されます。セットアップの支援、モニターリング、データ分析、または問題の検出には料金がかかりません。

CloudWatch Application Insights のコスト

Amazon EC2 のコストには、次の機能の使用が含まれます。

- CloudWatch エージェント
  - CloudWatch エージェントロググループ
  - CloudWatch エージェントメトリクス
  - Prometheus ロググループ (JMX ワークロード用)

すべてのリソースのコストには、次の機能の使用が含まれます。

- CloudWatch アラーム (大部分のコスト)
- SSM OpsItems (最低コスト)

コスト計算例

この例のコストは、次のシナリオに従って考慮されます。

次を含むリソースグループが作成されました。

- SQL Server がインストールされている Amazon EC2 インスタンス。
- アタッチされた Amazon EBS ボリューム。

このリソースグループに CloudWatch Application Insights をオンボードすると、Amazon EC2 インスタンスにインストールされている SQL Server ワークロードが検出されます。CloudWatch Application Insights は、次のメトリクスのモニターリングを開始します。

SQL Server インスタンスでは、次のメトリクスがモニターリングされます。

- CPUUtilization
- StatusCheckFailed
- Memory % Committed Bytes In Use
- Memory Available Mbytes
- Network Interface Bytes Total/sec
- Paging File % Usage
- Physical Disk % Disk Time
- Processor % Processor Time
- SQLServer:Buffer Manager cache hit ratio
- SQLServer:Buffer Manager life expectancy
- SQLServer:General Statistics Processes blocked
- SQLServer:General Statistics User Connections
- SQLServer:Locks Number of Deadlocks/sec
- SQLServer:SQL Statistics Batch Requests/sec
- System Processor Queue Length

SQL Server インスタンスにアタッチされたボリュームについては、以下のメトリクスがモニターリングされます。

- VolumeReadBytes
- VolumeWriteBytes
- VolumeReadOps
- VolumeWriteOps
- VolumeTotalReadTime
- VolumeTotalWriteTime
- VolumeIdleTime
- VolumeQueueLength
- VolumeThroughputPercentage
- VolumeConsumedReadWriteOps
- BurstBalance

このシナリオでは、コストは CloudWatch の料金ページと SSM の料金ページに従って計算されます。

- カスタムメトリクス

  このシナリオでは、CloudWatch エージェントを使用して、上記のメトリクスのうち 13 個が CloudWatch に発行されます。これらのメトリクスは、カスタムメトリクスとして処理されます。各カスタムメトリクスの月額コストは 0.3 USD です。これらのカスタムメトリクスの合計コストは、13 個 x 0.3 USD = 3.90 USD/月です。
- アラーム

  このシナリオでは、CloudWatch Application Insights は合計 26 個のメトリクスをモニターリングし、26 個のアラームが作成されます。各アラームの月額コストは 0.1 USD です。アラームの合計コストは、26 個 x 0.1 USD = 2.60 USD/月です。
- データインジェストとエラーログ

  データインジェストのコストは 1 GB あたり 0.05 USD で、SQL Server エラーログ用ストレージのコストは 1 GB あたり 0.03 USD です。データインジェストとエラーログの合計コストは、1 GB あたり 0.08 USD (0.05 USD + 0.03 USD) です。
- Amazon EC2 Systems Manager OpsItems

SSM OpsItem は、CloudWatch Application Insights が検出した問題ごとに作成されます。アプリケーションの問題が n 件発生した場合、合計月額コストは 0.00267 USD x n で求めることができます。

関連する のサービス

CloudWatch Application Insights は以下のサービスと併用されます。

関連 AWS サービス

- Amazon CloudWatch。リソースの使用状況、アプリケーションのパフォーマンス、および運用状態についてシステム全体の可視性を提供します。メトリクスの収集と追跡、アラーム通知の送信、定義したルールに基づくモニターリング対象リソースの自動更新を行います。また、ユーザー独自のカスタムメトリクスもモニターリングできます。CloudWatch Application Insights は、CloudWatch (具体的には CloudWatch のデフォルトのオペレーションダッシュボード内) から開始します。詳細については、Amazon CloudWatch ユーザーガイドを参照してください。
- CloudWatch Container Insights は、コンテナ化されたアプリケーションおよびマイクロサービスのメトリクスとログを収集、集約、要約します。Container Insights を使用して、Amazon EC2 の Kubernetes プラットフォーム、Amazon Elastic Kubernetes Service、Amazon ECS をモニターリングできます。Container Insights または Application Insights コンソールで Application Insights が有効になっている場合、Application Insights により、検出された問題が Container Insights ダッシュボードに表示されます。詳細については、「Container Insights」を参照してください。
- Amazon DynamoDB はフルマネージド型の NoSQL データベースサービスです。これを使用すると、分散データベースの運用とスケーリングに伴う管理作業をまかせることができるため、ハードウェアのプロビジョニング、設定と構成、レプリケーション、ソフトウェアパッチ適用、クラスタースケーリングなどを自分で行う必要はなくなります。また、DynamoDB も保管時の暗号化を提供し、機密データの保護における負担と複雑な作業を解消します。
- Amazon EC2 は、AWS クラウド内でスケーラブルなコンピューティング性能を提供します。Amazon EC2 を使用すると、必要な数 (またはそれ以下) の仮想サーバーを起動して、セキュリティおよびネットワーキングを構成し、ストレージを管理できます。要件の変更や需要増に対応してスケールアップまたはスケールダウンできるため、トラフィック予測を軽減できます。詳細については、Linux インスタンス用の Amazon EC2 ユーザーガイドまたは Windows インスタンス用の Amazon EC2 ガイドを参照してください。
- Amazon Elastic Block Store (Amazon EBS) は、Amazon EC2 インスタンスで使用するためのブロックレベルのストレージボリュームを提供します。Amazon EBS ボリュームの動作は、未初期化のブロックデバイスに似ています。これらのボリュームは、デバイスとしてインスタンスにマウ

ントできます。インスタンスにアタッチした Amazon EBS ボリュームは、インスタンスの有効期間とは無関係に存続するストレージボリュームとして公開されます。これらのボリューム上にファイルシステムを構築できます。または、これらのボリュームをブロックデバイス (ハードドライブなど) を使用する場合と同じ方法で使用できます。インスタンスにアタッチされているボリュームの設定は動的に変更できます。詳細は、Amazon EBS ユーザーガイドをご覧ください。

- Amazon EC2 Auto Scaling は、アプリケーションの負荷を処理するために適切な数の Amazon EC2 インスタンスを利用できるようにします。詳細については、Amazon EC2 Auto Scaling ユーザーガイドを参照してください。
- Elastic Load Balancing は、受信したアプリケーションまたはネットワークトラフィックを複数のアベイラビリティーゾーンの複数のターゲット (EC2 インスタンス、コンテナ、IP アドレスなど) に分散させます。詳細については、Elastic Load Balancing ユーザーガイドを参照してください。
- IAM は、AWS リソースへのユーザーアクセスを安全に管理するウェブサービスです。IAM により、どのユーザーがお客様の AWS リソースを使用できるか (認証)、それらのユーザーがどのリソースをどのような方法で使用できるか (承認) をコントロールできます。詳細については、「Amazon CloudWatch に対する認証とアクセスコントロール」を参照してください。
- AWS Lambda を使用すると、イベントによってトリガーされる関数で構成されたサーバーレスアプリケーションを構築し、これらを CodePipeline や AWS CodeBuild を使用して自動的にデプロイできます。詳細については、「AWS Lambda アプリケーション」をご参照ください。
- AWS Launch Wizard for SQL Server は、SQL Server 高可用性ソリューションをクラウドにデプロイする所要時間を短縮します。パフォーマンス、ノード数、サービスコンソールの接続を含むアプリケーションの要件を入力すると、AWS Launch Wizard は、SQL Server Always On アプリケーションをデプロイおよび実行するための適切な AWS リソースを識別します。
- AWSResource Groupsは、アプリケーションを構成するリソースを整理するのに役立ちます。Resource Groups を使用すると、大量のリソースのタスクを一度に管理および自動化できます。アプリケーションごとに登録できるリソースグループは 1 つだけです。詳細については、AWSResource Groups ユーザーガイドを参照してください。
- Amazon SQS。分散されたソフトウェアシステムとコンポーネントを統合および分離できる、安全性、耐久性、可用性に優れたホストされたキューを提供します。詳細については、Amazon SQS ユーザーガイドを参照してください。
- AWS Step Functions はサーバーレス関数作成ツールであり、AWS 関数を含むさまざまな AWS Lambda のサービスとリソースを、構造化された視覚的なワークフローに順序付けすることができます。詳細については、AWS Step Functions ユーザーガイドをご参照ください。
- AWS SSM OpsCenter では、各 OpsItem、関連する OpsItems、関連リソースに関するコンテキスト調査データを提供しながら、サービス全体で OpsItems を集約および標準化します。OpsCenter には Systems Manager Automation ドキュメント (ランブック) も用意されており、問題をす

ばやく解決できます。検索可能なカスタムデータを OpsItem ごとに指定することができます。OpsItems に関する自動的に生成された概要レポートは、ステータスおよびソース別に表示することもできます。詳細については、AWS Systems Manager ユーザーガイドを参照してください。

- Amazon API Gateway は、あらゆる規模の REST、HTTP、および WebSocket API を作成、公開、維持、モニターリング、およびセキュア化するための AWS サービスです。API 開発者は、AWS または他のウェブサービス、AWS クラウドに保存されているデータにアクセスする API を作成できます。詳細については、Amazon API Gateway ユーザーガイドを参照してください。

  > **Note**
  >
  > Application Insights は REST API プロトコル (API Gateway サービスの v1) のみをサポートします。

- Amazon Elastic Container Service (Amazon ECS) は、フルマネージド型のコンテナオーケストレーションサービスです。Amazon ECS を使用して、最も機密性の高いミッションクリティカルなアプリケーションを実行できます。詳細については、Amazon Elastic Container Service デベロッパーガイドを参照してください。
- Amazon Elastic Kubernetes Service (Amazon EKS) は、お客様独自の Kubernetes コントロールプレーンまたはノードをインストール、運用、保守管理することなく、AWS で Kubernetes を実行できるマネージドサービスです。Kubernetes は、コンテナ化されたアプリケーションのデプロイ、スケーリング、および管理を自動化するためのオープンソースシステムです。詳細については、Amazon EKS ユーザーガイドをご覧ください。
- Kubernetes on Amazon EC2。Kubernetes は、コンテナ化アプリケーションを大規模にデプロイおよび管理するのを支援するオープンソースソフトウェアです。Kubernetes では、デプロイ、メンテナンス、スケーリングのプロセスにより、Amazon EC2 コンピューティングインスタンスのクラスターを管理し、これらのインスタンスでコンテナを実行します。Kubernetes では、同じツールセットを使用して、オンプレミスやクラウド上で、すべてのタイプのコンテナ化アプリケーションを実行できます。詳細については、「Kubernetes ドキュメント：開始方法」を参照してください。
- Amazon FSxは、AWS によって完全に管理される一般的なファイルシステムの起動と実行に役立ちます。Amazon FSx を使用すると、一般的なオープンソースおよび商用ライセンスのファイルシステムの機能セットとパフォーマンスを活用して、時間のかかる管理タスクを回避できます。詳細については、「Amazon FSx のドキュメント」参照してください。
- Amazon Simple Notification Service (SNS) は、アプリケーション間およびアプリケーションと人間の通信の両方に対応するフルマネージド型のメッセージングサービスです。Amazon SNS

は、Application Insights によるモニターリング用に設定できます。Amazon SNS がモニターリング用のリソースとして設定されている場合、Application Insights は SNS のメトリクスを追跡して、SNS メッセージにおける問題の発生や失敗の理由を判定するのに役立ちます。

- Amazon Elastic File System (Amazon EFS) は、AWS クラウド のサービスやオンプレミスのリソースで使用できるフルマネージド型の伸縮性の高い NFS ファイルシステムです。アプリケーションを中断することなく、必要に応じてペタバイト規模まで拡張することを目的に構築されています。ファイルの追加や削除に応じて自動的に拡張/縮小されるため、拡張に対応するための容量のプロビジョニングや管理は不要になります。詳細については、Amazon Elastic File System のドキュメントを参照してください。

関連するサードパーティーのサービス

- Application Insights でモニターリングされるいくつかのワークロードとアプリケーションについては、Prometheus JMX Exporter は AWS Systems Manager Distributor を使用してインストールされるため、CloudWatch Application Insights は Java 固有のメトリクスを取得できます。Java アプリケーションのモニターリングを選択すると、Application Insights によって Prometheus JMX Exporter が自動的にインストールされます。

サポートされるアプリケーションコンポーネント

CloudWatch Application Insights は、リソースグループをスキャンしてアプリケーションコンポーネントを識別します。コンポーネントは、スタンドアロン、自動グループ化 (Auto Scaling グループ内のインスタンスやロードバランサーの背後のインスタンスなど)、またはカスタム (個々の Amazon EC2 インスタンスのグループ化) のいずれかになります。

CloudWatch Application Insights でサポートされるコンポーネントは以下のとおりです。

AWS コンポーネント

- Amazon EC2
- Amazon EBS
- Amazon RDS
- Elastic Load Balancing: Application Load Balancer と Classic Load Balancer (これらのロードバランサーのすべてのターゲットインスタンスが特定および設定されます)。
- Amazon EC2 Auto Scaling グループ: AWS Auto Scaling (Auto Scaling グループは、すべてのターゲットインスタンスに対して動的に設定されます。アプリケーションがスケールアップする

と、CloudWatch Application Insights は新しいインスタンスを自動的に設定します)。Auto Scaling グループは、CloudFormation のスタックベースのリソースグループではサポートされません。

- AWS Lambda
- Amazon Simple Queue Service (Amazon SQS)
- Amazon DynamoDB テーブル
- Amazon S3 バケットメトリクス
- AWS Step Functions
- Amazon API Gateway REST API ステージ
- Amazon Elastic Container Service (Amazon ECS): クラスター、サービス、およびタスク
- Amazon Elastic Kubernetes Service (Amazon EKS): クラスター
- Amazon EC2 での Kubernetes: EC2 で実行されている Kubernetes クラスター
- Amazon SNS トピック

その他すべてのコンポーネントタイプのリソースは、現在 CloudWatch Application Insights によって追跡されていません。サポートされているコンポーネントタイプが Application Insights アプリケーションに表示されない場合、そのコンポーネントは登録済みで、お客様が所有する別のアプリケーションで管理され、Application Insights によってモニターリングされている可能性があります。

サポートされるテクノロジースタック

CloudWatch Application Insights を使用して、以下のいずれかのテクノロジーについて、アプリケーション層ドロップダウンメニューオプションを選択すると、Windows Server および Linux オペレーティングシステムで実行されているアプリケーションをモニターリングできます。

- フロントエンド: Microsoft Internet Information Services (IIS) ウェブサーバー
- ワーカー層:
  - .NET Framework
  - .NET Core
- アプリケーション:
  - Java
  - SAP NetWeaver の標準、分散型、および高可用性デプロイ
- アクティブディレクトリ
- SharePoint

- データベース:
  - Amazon RDS または Amazon EC2 で実行されている Microsoft SQL Server (SQL Server High Availability の設定を含みます。コンポーネント設定の例 を参照してください。)
  - Amazon RDS、Amazon Aurora、Amazon EC2 上で実行されている MySQL
  - Amazon RDS または Amazon EC2 上で実行されている PostgreSQL
  - Amazon DynamoDB テーブル
  - Amazon RDS または Amazon EC2 上で実行されている Oracle
  - 単一の Amazon EC2 インスタンスおよび複数の EC2 インスタンス上の SAP HANA データベース
  - クロス AZ SAP HANA データベースの高可用性セットアップ。
  - 単一の Amazon EC2 インスタンス上のSAP Sybase ASE データベース。
  - クロス AZ SAP Sybase ASE データベースの高可用性セットアップ。

上に示したテクノロジースタックのいずれもアプリケーションリソースに該当しない場合は、[Manage Monitoring] ページのアプリケーション層のドロップダウンメニューから [Custom] を選択して、アプリケーションスタックをモニターリングできます。

Amazon CloudWatch Application Insights の仕組み

このセクションでは、CloudWatch Application Insights の仕組みについて説明します。これには以下が含まれます。

- Application Insights がアプリケーションをモニターリングする方法
- データ保持期間
- クォータ
- AWSCloudWatch Application Insights で使用される Systems Manager (SSM) パッケージ
- CloudWatch Application Insights で使用される AWS Systems Manager (SSM) ドキュメント

Application Insights がアプリケーションをモニターリングする方法

Application Insights は、アプリケーションを次のようにモニターリングします。

アプリケーションの検出と設定

CloudWatch Application Insights にアプリケーションを初めて追加すると、アプリケーションコンポーネントがスキャンされ、アプリケーションでモニターリングする主要なメトリクス、ログ、そ

の他のデータソースが推奨されます。これらの推奨内容に基づいて、アプリケーションを設定できます。

データの前処理

CloudWatch Application Insights は、アプリケーションリソース全体でモニターリングされているデータソースを継続的に分析し、メトリクスの異常やログエラー (監視結果) を検出します。

問題のインテリジェントな検出

CloudWatch Application Insights エンジンは、分類アルゴリズムと組み込みルールを使用して監視結果を関連付けることで、アプリケーションの問題を検出します。トラブルシューティングを支援するために CloudWatch ダッシュボードが自動的に作成され、問題に関するコンテキスト情報が表示されます。

アラートとアクション

CloudWatch Application Insights は、アプリケーションの問題を検出すると、CloudWatch Events を生成して問題をユーザーに通知します。これらのイベントをセットアップする方法の詳細については、「Application Insights CloudWatch Events と検出された問題の通知」を参照してください。

シナリオの例

SQL Server データベースを基盤とする ASP .NET アプリケーションがあります。メモリ負担が高いため、突然、データベースの誤動作が始まります。これに伴って、アプリケーションのパフォーマンスが低下し、ウェブサーバーやロードバランサーで HTTP 500 エラーが発生する可能性があります。

CloudWatch Application Insights とそのインテリジェントな分析により、動的に作成されるダッシュボードに関連メトリクスやログファイルのスニペットが表示されます。これを確認することで、問題を起こしているアプリケーション層を特定できます。この場合、問題は SQL データベース層にある可能性があります。

データ保持期間

CloudWatch Application Insights は、問題を 55 日間、監視結果を 60 日間保持します。

クォータ

CloudWatch Application Insights のデフォルトクォータについては、「Amazon CloudWatch Application Insights エンドポイントとクォータ」を参照してください。特に明記しない限り、

各クォータは AWS リージョン単位です。サービスのクォータの引き上げをリクエストする場合は、AWS Support にお問い合わせください。多くのサービスには、変更することができないクォータが含まれています。特定のサービスに対するクォータの詳細については、そのサービスのドキュメントを参照してください。

AWSCloudWatch Application Insights で使用される Systems Manager (SSM) パッケージ

このセクションに記載されているパッケージは Application Insights で使用されており、AWS Systems Manager Distributor で個別に管理およびデプロイできます。SSM Distributor の詳細については、AWS Systems Manager ユーザーガイドの AWS Systems Manager Distributor を参照してください。

パッケージ:

- AWSObservabilityExporter-JMXExporterInstallAndConfigure
- AWSObservabilityExporter-SAP-HANADBExporterInstallAndConfigure
- AWSObservabilityExporter-HAClusterExporterInstallAndConfigure
- AWSObservabilityExporter-SAP-SAPHostExporterInstallAndConfigure
- AWSObservabilityExporter-SQLExporterInstallAndConfigure

`AWSObservabilityExporter-JMXExporterInstallAndConfigure`

Application Insights の Prometheus JMX Exporter からワークロード固有の Java メトリクスを取得して、アラームを設定およびモニターリングできます。Application Insights コンソールの [Manage monitoring] (モニターリングの管理) ページで、[Application tier] (アプリケーション層) ドロップダウンから [JAVA application] (JAVA アプリケーション) を選択します。その後、[JAVA Prometheus exporter configuration] (JAVA Prometheus エクスポーター設定) で、[Collection method] (収集方法) と [JMX port number] (JMX ポート番号) を選択します。

AWS Systems Manager Distributor を使用して AWS が提供する Prometheus JMX Exporter パッケージを Application Insights とは別にパッケージ化、インストール、および設定するには、次の手順を実行します。

Prometheus JMX Exporter SSM パッケージを使用するための前提条件

- SSM エージェントバージョン 2.3.1550.0 以降がインストールされている
- JAVA_HOME 環境変数が設定されている

AWSObservabilityExporter-JMXExporterInstallAndConfigure パッケージをインストールして設定する

AWSObservabilityExporter-JMXExporterInstallAndConfigure パッケージは、Prometheus JMX Exporter のインストールおよび設定に使用できる SSM Distributor パッケージです。Prometheus JMX Exporter によって Java メトリクスが送信されると、CloudWatch サービスのメトリクスを取得するように CloudWatch エージェントを設定できます。

1. ご自身の好みに基づいて、Prometheus GitHub リポジトリにある Prometheus JMX Exporter の YAML 設定ファイルを準備します。設定例とオプションの説明を参考にしてください。
2. Base64 としてエンコードされた Prometheus JMX Exporter の YAML 設定ファイルを、SSM Parameter Store の新しい SSM パラメータにコピーします。
3. SSM Distributor コンソールに移動し、[Owned by Amazon] (Amazon 所有) タブを開きます。[AWSObservabilityExporter-JMXExporterInstallAndConfigure] を選択し、[Install one time] (1 回限りのインストール) を選択します。
4. 「Additional Arguments」を次のように置き換えて、最初のステップで作成した SSM パラメータを更新します。

```
{
  "SSM_EXPORTER_CONFIGURATION": "{{ssm:<SSM_PARAMETER_STORE_NAME>}}",
  "SSM_EXPOSITION_PORT": "9404"
}
```

> **Note**
>
> ポート 9404 は、Prometheus JMX メトリクスの送信に使用されるデフォルトのポートです。このポートは更新できます。

例: Java メトリクスを取得するための CloudWatch エージェントの設定

1. 前の手順で説明したように、Prometheus JMX Exporter をインストールします。その後、ポートのステータスをチェックして、インスタンスに正しくインストールされていることを確認します。

   Windows インスタンスでの正常なインストール例

```
PS C:\> curl http://localhost:9404 (http://localhost:9404/)
```

```
StatusCode : 200
StatusDescription : OK
Content : # HELP jvm_info JVM version info
```

Linux インスタンスでの正常なインストール例

```
$ curl localhost:9404
# HELP jmx_config_reload_failure_total Number of times configuration have failed to
 be reloaded.
# TYPE jmx_config_reload_failure_total counter
jmx_config_reload_failure_total 0.0
```

2. Prometheus サービスの検出 YAML ファイルを作成します。次のサンプルのサービス検出ファイルは、次の処理を実行します。

   - Prometheus JMX Exporter ホストポートを `localhost: 9404` として指定します。
   - メトリクスにラベル (`Application`、`ComponentName`、および `InstanceId`) をアタッチします。これは、CloudWatch メトリクスディメンションとして設定できます。

```
$ cat prometheus_sd_jmx.yaml
- targets:
  - 127.0.0.1:9404
  labels:
    Application: myApp
    ComponentName: arn:aws:elasticloadbalancing:us-
east-1:123456789012:loadbalancer/app/sampl-Appli-MMZW8E3GH4H2/aac36d7fea2a6e5b
    InstanceId: i-12345678901234567
```

3. Prometheus JMX Exporter 設定 YAML ファイルを作成します。次の設定ファイルの例では、次の項目を指定します。

   - メトリクスの取得ジョブ間隔とタイムアウト期間。
   - スクレイピングとも呼ばれるメトリクスの取得ジョブ (`jmx` および `sap`)。これには、ジョブ名、一度に返される最大時系列、およびサービス検出のファイルパスが含まれます。

```
$ cat prometheus.yaml
global:
  scrape_interval: 1m
  scrape_timeout: 10s
```

```
scrape_configs:
  - job_name: jmx
    sample_limit: 10000
    file_sd_configs:
      - files: ["/tmp/prometheus_sd_jmx.yaml"]
  - job_name: sap
    sample_limit: 10000
    file_sd_configs:
      - files: ["/tmp/prometheus_sd_sap.yaml"]
```

4. CloudWatch エージェントが Amazon EC2 インスタンスにインストールされ、バージョンが 1.247346.1b249759 以降であることを検証します。EC2 インスタンスに CloudWatch エージェントをインストールするには、「CloudWatch エージェントのインストール」を参照してください。バージョンを確認するには、「CloudWatch エージェントのバージョンについての情報の検索」をご参照ください。
5. CloudWatch エージェントを設定します。CloudWatch エージェント設定ファイルの設定方法の詳細については、「CloudWatch エージェント設定ファイルを手動で作成または編集する」を参照してください。次の CloudWatch エージェント設定ファイルの例は、次の処理を実行します。

   - Prometheus JMX Exporter 設定ファイルパスを指定します。
   - EMF メトリクスログを発行するターゲットロググループを指定します。
   - メトリクス名ごとに 2 つのディメンションセットを指定します。
   - 8 個 (4 個のメトリクス名 * メトリクス名ごとに 2 セットのディメンション) の CloudWatch メトリクスを送信します。

```
{
   "logs":{
      "logs_collected":{
         ....
      },
      "metrics_collected":{
         "prometheus":{
            "cluster_name":"prometheus-test-cluster",
            "log_group_name":"prometheus-test",
            "prometheus_config_path":"/tmp/prometheus.yaml",
            "emf_processor":{
               "metric_declaration_dedup":true,
               "metric_namespace":"CWAgent",
               "metric_unit":{
                  "jvm_threads_current":"Count",
```

```
                    "jvm_gc_collection_seconds_sum":"Second",
                    "jvm_memory_bytes_used":"Bytes"
                },
                "metric_declaration":[
                    {
                        "source_labels":[
                            "job"
                        ],
                        "label_matcher":"^jmx$",
                        "dimensions":[
                            [
                                "InstanceId",
                                "ComponentName"
                            ],
                            [
                                "ComponentName"
                            ]
                        ],
                        "metric_selectors":[
                            "^java_lang_threading_threadcount$",
                            "^java_lang_memory_heapmemoryusage_used$",
                            "^java_lang_memory_heapmemoryusage_committed$"
                        ]
                    }
                ]
            }
        }
    }
    },
    "metrics":{
        ....
    }
}
```

AWSObservabilityExporter-SAP-HANADBExporterInstallAndConfigure

Application Insights の Prometheus HANA データベースエクスポーター からワークロード固有の SAP HANA メトリクスを取得して、アラームを設定およびモニターリングできます。詳細については、このガイドの「SAP HANA データベースをモニターリング用に設定する」を参照してください。

AWS Systems Manager Distributor を使用して、AWS が提供する Prometheus HANA データベースエクスポーターパッケージを Application Insights とは別にパッケージ化、インストール、設定するには、次の手順を完了します。

Prometheus HANA データベース Exporter SSM パッケージを使用するための前提条件

- SSM エージェントバージョン 2.3.1550.0 以降がインストールされている
- SAP HANA データベース
- Linux オペレーティングシステム (SUSE Linux、RedHat Linux)
- AWS Secrets Manager を使用しており、SAP HANA データベースのモニタリング認証情報を持つシークレット。キーと値のペアの形式を使用してシークレットを作成し、キーのユーザー名を指定して、値にデータベースユーザーを入力します。2 番目のキーのパスワードを追加して、値にパスワードを入力します。シークレットを作成する方法については、AWS Secrets Manager ユーザーガイドの「シークレットを作成する」を参照してください。シークレットは、次のようにフォーマットする必要があります。

```
{
  "username": "<database_user>",
  "password": "<database_password>"
}
```

AWSObservabilityExporter-SAP-HANADBExporterInstallAndConfigure パッケージをインストールして設定する

AWSObservabilityExporter-SAP-HANADBExporterInstallAndConfigure パッケージは、Prometheus HANA データベース Exporter のインストールおよび設定に使用できる SSM Distributor パッケージです。Prometheus HANA データベース Exporter によって HANA データベースのメトリクスが送信されると、CloudWatch サービスのメトリクスを取得するように CloudWatch エージェントを設定できます。

1. SSM パラメータストアで SSM パラメータを作成して、Exporter の設定を保存します。パラメータ値の例を次に示します。

```
{\"exposition_port\":9668,\"multi_tenant\":true,\"timeout\":600,\"hana\":{\"host\":
\"localhost\",\"port\":30013,\"aws_secret_name\":\"HANA_DB_CREDS\",\"scale_out_mode
\":true}}
```

> **Note**
>
> この例では、エクスポートはアクティブな SYSTEM データベースがある Amazon EC2 インスタンスでのみ実行されます。また、メトリクスの重複を避けるため、他の EC2 インスタンスはアイドル状態のままになります。エクスポーターでは、SYSTEM データベースからのすべてのテナント情報を取得できます。

2. SSM パラメータストアで SSM パラメータを作成して、Exporter メトリクスのクエリを保存します。パッケージでは、複数のメトリクスパラメータが受け入れ可能です。各パラメータには、有効な JSON オブジェクトの形式が必要です。パラメータの値の例を次に示します。

```
{\"SELECT MAX(TIMESTAMP) TIMESTAMP, HOST, MEASURED_ELEMENT_NAME CORE,
 SUM(MAP(CAPTION, 'User Time', TO_NUMBER(VALUE), 0)) USER_PCT, SUM(MAP(CAPTION,
 'System Time', TO_NUMBER(VALUE), 0)) SYSTEM_PCT, SUM(MAP(CAPTION, 'Wait
 Time', TO_NUMBER(VALUE), 0)) WAITIO_PCT, SUM(MAP(CAPTION, 'Idle Time', 0,
 TO_NUMBER(VALUE))) BUSY_PCT, SUM(MAP(CAPTION, 'Idle Time', TO_NUMBER(VALUE), 0))
 IDLE_PCT FROM sys.M_HOST_AGENT_METRICS WHERE MEASURED_ELEMENT_TYPE = 'Processor'
 GROUP BY HOST, MEASURED_ELEMENT_NAME;\":{\"enabled\":true,\"metrics\":[{\"name\":
\"hanadb_cpu_user\",\"description\":\"Percentage of CPU time spent by HANA DB in user
 space, over the last minute (in seconds)\",\"labels\":[\"HOST\",\"CORE\"],\"value\":
\"USER_PCT\",\"unit\":\"percent\",\"type\":\"gauge\"},{\"name\":\"hanadb_cpu_system
\",\"description\":\"Percentage of CPU time spent by HANA DB in Kernel space,
 over the last minute (in seconds)\",\"labels\":[\"HOST\",\"CORE\"],\"value\":
\"SYSTEM_PCT\",\"unit\":\"percent\",\"type\":\"gauge\"},{\"name\":\"hanadb_cpu_waitio
\",\"description\":\"Percentage of CPU time spent by HANA DB in IO mode, over the
 last minute (in seconds)\",\"labels\":[\"HOST\",\"CORE\"],\"value\":\"WAITIO_PCT\",
\"unit\":\"percent\",\"type\":\"gauge\"},{\"name\":\"hanadb_cpu_busy\",\"description
\":\"Percentage of CPU time spent by HANA DB, over the last minute (in seconds)\",
\"labels\":[\"HOST\",\"CORE\"],\"value\":\"BUSY_PCT\",\"unit\":\"percent\",\"type\":
\"gauge\"},{\"name\":\"hanadb_cpu_idle\",\"description\":\"Percentage of CPU time not
 spent by HANA DB, over the last minute (in seconds)\",\"labels\":[\"HOST\",\"CORE
\"],\"value\":\"IDLE_PCT\",\"unit\":\"percent\",\"type\":\"gauge\"}]}}
```

メトリクスのクエリについての詳細は、GitHub の「SUSE / hanadb_exporter repo」を参照してください。

3. SSM Distributor コンソールに移動し、[Owned by Amazon] (Amazon 所有) タブを開きます。[AWSObservabilityExporter-SAP-HANADBExporterInstallAndConfigure*] を選択し、[Install one time] (1 回限りのインストール) を選択します。

4. 「Additional Arguments」を次のように置き換えて、最初のステップで作成した SSM パラメータを更新します。

```
{
  "SSM_EXPORTER_CONFIG": "{{ssm:<*SSM_CONFIGURATIONS_PARAMETER_STORE_NAME>*}}",
  "SSM_SID": "<SAP_DATABASE_SID>",
  "SSM_EXPORTER_METRICS_1": "{{ssm:<SSM_FIRST_METRICS_PARAMETER_STORE_NAME>}}",
  "SSM_EXPORTER_METRICS_2": "{{ssm:<SSM_SECOND_METRICS_PARAMETER_STORE_NAME>}}"
}
```

5. SAP HANA データベースを持つ Amazon EC2 インスタンスを選択し、[Run] (実行) をクリックします。

AWSObservabilityExporter-HAClusterExporterInstallAndConfigure

Prometheus HANA クラスターエクスポーター から Application Insights 用に ワークロード固有の High Availability (HA) クラスターメトリクスを取得して、SAP HANA データベースの High Availability セットアップ用のアラームを設定およびモニターリングできます。詳細については、このガイドの「SAP HANA データベースをモニターリング用に設定する」を参照してください。

AWS Systems Manager Distributor を使用して、AWS が提供する Prometheus HA クラスター Exporter のパッケージを Application Insights とは別にパッケージ化、インストール、設定するには、次の手順を完了します。

Prometheus HA クラスター Exporter SSM パッケージを使用するための前提条件

- SSM エージェントバージョン 2.3.1550.0 以降がインストールされている
- Pacemaker、Corosync、SBD、DRBD 用の HA クラスター
- Linux オペレーティングシステム (SUSE Linux、RedHat Linux)

AWSObservabilityExporter-HAClusterExporterInstallAndConfigure パッケージをインストールして設定する

AWSObservabilityExporter-HAClusterExporterInstallAndConfigure パッケージは、Prometheus HA クラスター Exporter のインストールおよび設定に使用できる SSM Distributor のパッケージです。Prometheus HANA データベース Exporter によってクラスターメトリクスが送信されると、CloudWatch サービスのメトリクスを取得するように CloudWatch エージェントを設定できます。

1. SSM パラメータストアで SSM パラメータを作成して、Exporter の設定を JSON 形式で保存します。パラメータ値の例を次に示します。

```
{\"port\":\"9664\",\"address\":\"0.0.0.0\",\"log-level\":\"info\",\"crm-mon-path
\":\"/usr/sbin/crm_mon\",\"cibadmin-path\":\"/usr/sbin/cibadmin\",\"corosync-
cfgtoolpath-path\":\"/usr/sbin/corosync-cfgtool\",\"corosync-quorumtool-path\":\"/
usr/sbin/corosync-quorumtool\",\"sbd-path\":\"/usr/sbin/sbd\",\"sbd-config-path\":
\"/etc/sysconfig/sbd\",\"drbdsetup-path\":\"/sbin/drbdsetup\",\"enable-timestamps
\":false}
```

   エクスポーターの設定についての詳細は、GitHub の「ClusterLabs / ha_cluster_exporter repo」を参照してください。
2. SSM Distributor コンソールに移動し、[Owned by Amazon] (Amazon 所有) タブを開きます。[AWSObservabilityExporter-HAClusterExporterInstallAndConfigure*] を選択し、[Install one time] (1 回限りのインストール) を選択します。
3. 「Additional Arguments」を次のように置き換えて、最初のステップで作成した SSM パラメータを更新します。

```
{
  "SSM_EXPORTER_CONFIG": "{{ssm:<*SSM_CONFIGURATIONS_PARAMETER_STORE_NAME>*}}"
}
```

4. SAP HANA データベースを持つ Amazon EC2 インスタンスを選択し、[Run] (実行) をクリックします。

AWSObservabilityExporter-SAP-SAPHostExporterInstallAndConfigure

Application Insights の Prometheus SAP ホストエクスポーターからワークロード固有の SAP NetWeaver メトリクスを取得して、SAP NetWeaver Distributed and High Availability デプロイ用のアラームを設定およびモニターリングできます。詳細については、「Amazon CloudWatch Application Insights の開始方法」を参照してください。

AWS Systems Manager Distributor を使用して、SAP ホストエクスポーターパッケージを Application Insights とは別にパッケージ化、インストール、および設定するには、次の手順を実行します。

Prometheus SAP ホストエクスポーター SSM パッケージを使用するための前提条件

- SSM エージェントバージョン 2.3.1550.0 以降がインストールされている

- SAP NetWeaver アプリケーションサーバー
- Linux オペレーティングシステム (SUSE Linux、RedHat Linux)

`AWSObservabilityExporter-SAP-SAPHostExporterInstallAndConfigure` パッケージをインストールして設定する

`AWSObservabilityExporter-SAP-SAPHostExporterInstallAndConfigure` パッケージは、SAP NetWeaver Prometheus メトリクスエクスポーターのインストールおよび設定に使用できる SSM Distributor パッケージです。Prometheus エクスポーターによって SAP NetWeaver メトリクスが送信されると、CloudWatch サービスのメトリクスを取得するように CloudWatch エージェントを設定できます。

1. SSM パラメータストアで SSM パラメータを作成して、Exporter の設定を JSON 形式で保存します。パラメータ値の例を次に示します。

```
{\"address\":\"0.0.0.0\",\"port\":\"9680\",\"log-level\":\"info\",\"is-HA\":false}
```

   - address

     Prometheus メトリクスの送信先となるターゲットアドレス。デフォルト値は `localhost` です。

   - port

     Prometheus メトリクスの送信先となるターゲットポート。デフォルト値は `9680` です。

   - is-HA

     SAP NetWeaver High Availability のデプロイでは [`true`]。他のすべてのデプロイで、値は `false` です。

2. SSM Distributor コンソールに移動し、[Owned by Amazon] (Amazon 所有) タブを開きます。AWSObservabilityExporter-SAP-SAPHostExporterInstallAndConfigure を選択し、[Install one time] (1 回限りのインストール) を選択します。
3. 「Additional Arguments」を次のように置き換えて、最初のステップで作成した SSM パラメータを更新します。

```
{
  "SSM_EXPORTER_CONFIG": "{{ssm:<SSM_CONFIGURATIONS_PARAMETER_STORE_NAME>}}",
  "SSM_SID": "<SAP_DATABASE_SID>",
  "SSM_INSTANCES_NUM": "<instances_number seperated by comma>"
```

```
}
```

例

```
{
  "SSM_EXPORTER_CONFIG": "{{ssm:exporter_config_paramter}}",
  "SSM_INSTANCES_NUM": "11,12,10",
  "SSM_SID": "PR1"
}
```

4. SAP NetWeaver アプリケーションを含む Amazon EC2 インスタンスを選択し、[Run] (実行) を選択します。

> **Note**
>
> Prometheus エクスポーターは、ローカルエンドポイントで SAP NetWeaver メトリクスを処理します。ローカルエンドポイントには Amazon EC2 インスタンスのオペレーティングシステムユーザーのみがアクセスできます。そのため、エクスポーターパッケージをインストールすると、すべてのオペレーティングシステムユーザーがメトリクスを利用できるようになります。デフォルトのローカルエンドポイントは `localhost:9680/metrics` です。

`AWSObservabilityExporter-SQLExporterInstallAndConfigure`

Application Insights の Prometheus SQL エクスポーターからワークロード固有の SQL Server メトリクスを取得して、主要なメトリクスをモニタリングできます。

AWS Systems Manager Distributor を使用して、SQL エクスポーターパッケージを Application Insights とは別にパッケージ化、インストール、および設定するには、次のステップを実行します。

Prometheus SQL エクスポーター SSM パッケージを使用するための前提条件

- SSM エージェントバージョン 2.3.1550.0 以降がインストールされている
- SQL Server ユーザー認証が有効になっている Windows 上で SQL Server を実行している Amazon EC2 インスタンス。
- 次の許可を持つ SQL Server ユーザー:

```
GRANT VIEW ANY DEFINITION TO
```

```
GRANT VIEW SERVER STATE TO
```

- AWS Secrets Manager を使用したデータベース接続文字列を含むシークレット。シークレットを作成する方法については、AWS Secrets Manager ユーザーガイドの「シークレットを作成する」を参照してください。シークレットは、次のようにフォーマットする必要があります。

```
{
  "data_source_name":"sqlserver://<username>:<password>@localhost:1433"
}
```

> **Note**
>
> パスワードまたはユーザー名に特殊文字が含まれている場合は、データベースに正常に接続できるように特殊文字をパーセントエンコーディングする必要があります。

AWSObservabilityExporter-SQLExporterInstallAndConfigure パッケージをインストールして設定する

AWSObservabilityExporter-SQLExporterInstallAndConfigure パッケージは、SQL Prometheus メトリクスエクスポーターのインストールおよび設定に使用できる SSM Distributor パッケージです。Prometheus エクスポーターによってメトリクスが送信されると、CloudWatch サービスのメトリクスを取得するように CloudWatch エージェントを設定できます。

1. 任意で、SQL エクスポーター YAML 設定を準備します。次のサンプル設定では、単一のメトリクスが設定されています。設定例を使用して、追加のメトリクスで設定を更新するか、または独自の設定を作成します。

```
---
global:
  scrape_timeout_offset: 500ms
  min_interval: 0s
  max_connections: 3
  max_idle_connections: 3
target:
  aws_secret_name: <SECRET_NAME>
  collectors:
    - mssql_standard
collectors:
```

```
  - collector_name: mssql_standard
    metrics:
    - metric_name: mssql_batch_requests
      type: counter
      help: 'Number of command batches received.'
      values: [cntr_value]
      query: |
        SELECT cntr_value
        FROM sys.dm_os_performance_counters WITH (NOLOCK)
        WHERE counter_name = 'Batch Requests/sec'
```

2. Base64 としてエンコードされた Prometheus SQL エクスポーターの YAML 設定ファイルを、SSM Parameter Store の新しい SSM パラメータにコピーします。
3. SSM Distributor コンソールに移動し、[Owned by Amazon] (Amazon 所有) タブを開きます。AWSObservabilityExporter-SQLExporterInstallAndConfigure を選択し、[1 回限りのインストール] を選択します。
4. 「Additional Arguments」を次の情報に置き換えます。SSM_PARAMETER_NAME は、ステップ 2 で作成したパラメータの名前です。

```
{
  "SSM_EXPORTER_CONFIGURATION":
    "{{ssm:<SSM_PARAMETER_STORE_NAME>}}",
    "SSM_PROMETHEUS_PORT": "9399",
    "SSM_WORKLOAD_NAME": "SQL"
}
```

5. SQL Server データベースを含む Amazon EC2 インスタンスを選択し、[実行] を選択します。

CloudWatch Application Insights で使用される AWS Systems Manager (SSM) ドキュメント

Application Insights は、このセクションに記載されている SSM ドキュメントを使用して、AWS Systems Manager がマネージドインスタンスで実行するアクションを定義します。これらのドキュメントでは、Systems Manager の Run Command 機能を使用して、Application Insights のモニタリング機能の実行に必要なタスクを自動化しています。これらのドキュメントの実行スケジュールは、Application Insights によって管理されており、変更することはできません。

SSM ドキュメントの詳細については、「AWS Systems Manager ユーザーガイド」の「AWS Systems Manager ドキュメント」を参照してください。

CloudWatch Application Insights によって管理されるドキュメント

次の表は、Application Insights によって管理される SSM ドキュメントの一覧です。

ドキュメント名	説明	実行スケジュール
AWSEC2-DetectWorkload	Application Insights によるモニタリング対象として設定できるアプリケーション環境で実行されているアプリケーションを自動検出します。	このドキュメントはアプリケーション環境で 1 時間ごとに実行され、アプリケーションに関する最新の詳細情報を取得します。
AWSEC2-CheckPerformanceCounterSets	Amazon EC2 Windows インスタンスでパフォーマンスカウンターの名前空間が有効になっているかどうかを確認します。	このドキュメントはアプリケーション環境で 1 時間ごとに実行され、対応する名前空間が有効になっている場合にのみパフォーマンスカウンターのメトリクスをモニタリングします。
AWSEC2-ApplicationInsightsCloudwatchAgentInstallAndConfigure	アプリケーションコンポーネントのモニタリング設定に基づいて、CloudWatch エージェントをインストールして設定します。	このドキュメントは 30 分ごとに実行され、CloudWatch エージェントの設定が常に正確で最新の状態になるようにします。また、このドキュメントは、メトリクスの追加や削除、ログ設定の更新など、アプリケーションのモニタリング設定に変更が加えられた直後にも実行されます。

AWS Systems Manager によって管理されるドキュメント

次のドキュメントは、CloudWatch Application Insights によって使用され、Systems Manager によって管理されます。

AWS-ConfigureAWSPackage

Application Insights はこのドキュメントを使用して、Prometheus エクスポーターディストリビューターパッケージのインストールとアンインストールを行い、ワークロード固有のメトリクスを収集し、顧客の Amazon EC2 インスタンスでワークロードを包括的にモニタリングできるようにします。CloudWatch Application Insights は、相関関係のあるターゲットワークロードがインスタンスで実行されている場合にのみ、Prometheus エクスポーターディストリビューターパッケージをインストールします。

次の表は、Prometheus エクスポーターディストリビューターパッケージと、相関関係のあるターゲットワークロードの一覧です。

Prometheus エクスポーターディストリビューターパッケージ名	ターゲットワークロード
`AWSObservabilityExporter-HAClusterExporterInstallAndConfigure`	SAP HANA HA
`AWSObservabilityExporter-JMXExporterInstallAndConfigure`	Java/JMX
`AWSObservabilityExporter-SAP-HANADBExporterInstallAndConfigure`	SAP HANA
`AWSObservabilityExporter-SAP-SAPHostExporterInstallAndConfigure`	NetWeaver
`AWSObservabilityExporter-SQLExporterInstallAndConfigure`	SQL サーバー (Windows) と SAP ASE (Linux)

`AmazonCloudWatch-ManageAgent`

Application Insights はこのドキュメントを使用して、インスタンスで CloudWatch エージェントのステータスと設定を管理し、オペレーティングシステム全体で Amazon EC2 インスタンスから内部システムレベルのメトリクスとログを収集します。

Amazon CloudWatch Application Insights の開始方法

CloudWatch Application Insights の使用を開始するには、次の前提条件を満たしていること、および IAM ポリシーが作成済みであることを確認します。次に、コンソールのリンクを使用して CloudWatch Application Insights を有効にすることで開始できます。アプリケーションのリソースを設定するには、「モニターリング用のアプリケーションのセットアップ、設定、管理」の手順に従います。

目次

CloudWatch Application Insights へのアクセス

次のいずれかのインターフェイスを使用して、CloudWatch Application Insights にアクセスしたり管理できます。

- CloudWatch コンソール。アプリケーションにモニターリングを追加するには、CloudWatch コンソールの左側のナビゲーションペインで [Insights] の下にある [Application Insights] を選択します。アプリケーションの設定が完了したら、 CloudWatch コンソールを使用して検出された問題を表示して分析できます。
- AWS Command Line Interface (AWS CLI)。AWS CLI を使用して、AWS API オペレーションにアクセスできます。詳細については、AWS Command Line Interface ユーザーガイドの AWS Command Line Interface のインストールを参照してください。Application Insights での API オペレーションについての詳細は、Amazon CloudWatch Application Insights API リファレンスを参照してください。

前提条件

CloudWatch Application Insights でアプリケーションを設定するには、以下の前提条件を満たす必要があります。

- AWS Systems Manager の有効化 – Amazon EC2 インスタンスに Systems Manager Agent (SSM Agent) をインストールし、インスタンスを SSM に対して有効にします。SSM Agent のインストール方法についての詳細は、「AWS Systems Manager ユーザーガイド」の「AWS Systems Manager のセットアップ」を参照してください。
- EC2 インスタンスロール – Systems Manager を有効にするには、次の Amazon EC2 インスタンスロールをアタッチする必要があります。
  - Systems Manager を有効にする `AmazonSSMManagedInstanceCore` ロールをアタッチする必要があります。詳細については、「AWS Systems Manager アイデンティティベースのポリシーの例」を参照してください。
  - インスタンスメトリクスとログが CloudWatch を介して出力されるように `CloudWatchAgentServerPolicy` ポリシーをアタッチする必要があります。詳細については、「CloudWatch エージェントで使用する IAM ロールとユーザーを作成する」を参照してください
- AWS リソースグループ – アプリケーションを CloudWatch Application Insights にオンボードするには、アプリケーションスタックで使用されるすべての関連する AWS リソースが含まれるリソースグループを作成する必要があります。これには、Application Load Balancer、IIS とウェブフロントエンドを実行する Amazon EC2 インスタンス、.NET ワーカー階層、および SQL Server データベースが含まれます。Application Insights がサポートするアプリケーションコンポーネントとテクノロジースタックの詳細については、「サポートされるアプリケーションコンポーネント」を参照してください。CloudWatch Application Insights では、同じタグを使用する Auto Scaling グループまたは CloudFormation スタックがリソースグループとして自動的に包含されます。Auto Scaling グループは CloudFormation リソースグループでサポートされていないためです。詳細については、「AWS Resource Groups の開始方法」を参照してください。
- IAM アクセス許可 – 管理アクセスが許可されていないユーザーについては、Application Insights でサービスにリンクされたロールを作成できる AWS Identity and Access Management (IAM) ポリシーを作成し、ユーザーの ID にアタッチする必要があります。IAM ポリシーを作成する方法については、「IAM ポリシー」を参照してください。
- サービスにリンクされたロール – Application Insights は AWS Identity and Access Management (IAM) サービスにリンクされたロールを使用します。Application Insights コンソールで新しい Application Insights アプリケーションを作成する際、サービスにリンクされたロールが作成されます。詳細については、「CloudWatch Application Insights のサービスにリンクされたロールの使用」を参照してください。
- EC2 Windows インスタンスに対するパフォーマンスカウンタメトリクスのサポート – Amazon EC2 Windows インスタンスでパフォーマンスカウンタメトリクスをモニターリングするには、パフォーマンスカウンタがインスタンスにインストールされている必要があります。パフォーマン

スカウンタのメトリクスと対応するパフォーマンスカウンタセット名については、「パフォーマンスカウンタのメトリクス」を参照してください。パフォーマンスカウンタの詳細については、「パフォーマンスカウンタ」を参照してください。

- Amazon CloudWatch エージェント – Application Insights は、CloudWatch エージェントをインストールして設定します。CloudWatch エージェントがインストールされている場合、Application Insights は設定をそのまま維持します。マージ競合を避けるため、既存の CloudWatch エージェント設定ファイルから、Application Insights で使用するリソースの設定を削除してください。詳細については、「 CloudWatch エージェント設定ファイルを手動で作成または編集する」を参照してください。

IAM ポリシー

CloudWatch Application Insights を使用するには、AWS Identity and Access Management (IAM) ポリシーを作成し、ユーザー、グループ、またはロールにアタッチする必要があります。ユーザー、グループ、ロールの詳細については、「IAM ID (ユーザー、ユーザーグループ、ロール)」を参照してください。IAM ポリシーは、ユーザーのアクセス許可を定義します。

コンソールを使用して IAM ポリシーを作成するには

IAM コンソールを使用して IAM ポリシーを作成するには、次の手順を実行します。

1. IAM コンソールに移動します。左のナビゲーションペインで [ポリシー] を選択します。
2. ページの上部で [ポリシーの作成] を選択します。
3. [JSON] タブを選択します。
4. 次の JSON ドキュメントをコピーして、[JSON] タブの下に貼り付けます。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "applicationinsights:*",
                "iam:CreateServiceLinkedRole",
                "iam:ListRoles",
                "resource-groups:ListGroups"
            ],
            "Effect": "Allow",
            "Resource": "*"
        }
```

```
    ]
}
```

5. [ポリシーの確認] をクリックします。
6. [名前] にポリシー名を入力します (「AppInsightsPolicy」など)。必要に応じて [説明] に説明を入力します。
7. [ポリシーの作成] を選択します。
8. 左側にあるナビゲーションペインで、[ユーザーグループ]、[ユーザー]、または [ロール] を選択します。
9. ポリシーのアタッチ先とするユーザー名、グループ名、ロール名を選択します。
10. [Add permissions (アクセス許可の追加)] を選択します。
11. [Attach existing policies directly (既存のポリシーを直接アタッチ)] を選択します。
12. 先ほど作成したポリシーを検索し、ポリシー名の左側にあるチェックボックスをオンにします。
13. 次に、[次へ: 確認] を選択します。
14. 適切なポリシーが表示されていることを確認し、 [Add permissions (アクセス許可の追加)] を選択します。
15. CloudWatch Application Insights を使用する際は、先ほど作成したポリシーを関連付けたユーザーとしてログインします。

AWS CLI を使用して IAM ポリシーを作成するには

AWS CLI を使用して IAM ポリシーを作成するには、上記の JSON ドキュメントを現在のフォルダ内のファイルとして使用して、コマンドラインから create-policy オペレーションを実行します。

AWS Tools for Windows PowerShell を使用して IAM ポリシーを作成するには

AWS Tools for Windows PowerShell を使用して IAM ポリシーを作成するには、上記の JSON ドキュメントを現在のフォルダ内のファイルとして使用して New-IAMPolicy cmdlt を実行します。

アカウントベースのアプリケーションのオンボーディングに対する IAM ロールのアクセス許可

アカウント内のすべてのリソースをオンボードし、Application Insights の機能にフルアクセスするための Application Insights 管理ポリシーを使用しない場合、Application Insights がアカウント内のすべてのリソースを検出できるように、IAM ロールに次のアクセス許可をアタッチする必要があります。

```
"ec2:DescribeInstances"
"ec2:DescribeNatGateways"
"ec2:DescribeVolumes"
"ec2:DescribeVPCs"
"rds:DescribeDBInstances"
"rds:DescribeDBClusters"
"sqs:ListQueues"
"elasticloadbalancing:DescribeLoadBalancers"
"autoscaling:DescribeAutoScalingGroups"
"lambda:ListFunctions"
"dynamodb:ListTables"
"s3:ListAllMyBuckets"
"sns:ListTopics"
"states:ListStateMachines"
"apigateway:GET"
"ecs:ListClusters"
"ecs:DescribeTaskDefinition"
"ecs:ListServices"
"ecs:ListTasks"
"eks:ListClusters"
"eks:ListNodegroups"
"fsx:DescribeFileSystems"
"route53:ListHealthChecks"
"route53:ListHostedZones"
"route53:ListQueryLoggingConfigs"
"route53resolver:ListFirewallRuleGroups"
"route53resolver:ListFirewallRuleGroupAssociations"
"route53resolver:ListResolverEndpoints"
"route53resolver:ListResolverQueryLogConfigs"
"route53resolver:ListResolverQueryLogConfigAssociations"
"logs:DescribeLogGroups"
"resource-explorer:ListResources"
```

モニターリング用のアプリケーションのセットアップ、設定、管理

このセクションでは、コンソール、AWS CLI、AWS Tools for Windows PowerShell を使用して、CloudWatch Application Insights アプリケーションをセットアップ、設定、管理する手順を説明します。

トピック

- CloudWatch コンソールからモニターリングするアプリケーションをセットアップ、設定、管理する

- コマンドラインを使用してモニターリングするアプリケーションをセットアップ、設定、管理する
- Application Insights CloudWatch Events と検出された問題の通知

CloudWatch コンソールからモニターリングするアプリケーションをセットアップ、設定、管理する

このセクションでは、CloudWatch コンソールからモニターリングするアプリケーションをセットアップ、設定、管理する手順を説明します。

コンソール手順

- アプリケーションの追加と設定
- Amazon ECS および Amazon EKS リソースのモニターリングで Application Insights を有効にする
- アプリケーションコンポーネントのモニターリングの無効化
- アプリケーションの削除

アプリケーションの追加と設定

CloudWatch コンソールからアプリケーションを追加および設定する

CloudWatch コンソールから CloudWatch Application Insights の使用を開始するには、次の手順を実行します。

1. 起動します。CloudWatch コンソールのランディングページを開きます。左側のナビゲーションペインで、[Insights] の下にある [Application Insights] を選択します。ページには、CloudWatch Application Insights でモニターリングするアプリケーションのリストと、モニターリングのステータスが表示されます。
2. アプリケーションを追加します。アプリケーションのモニターリングを設定するには、[Add an application] (アプリケーションの追加) を選択します。[Add an application] (アプリケーションの追加) を選択すると、[Choose Application Type] (アプリケーションタイプを選択してください) というプロンプトが表示されます。

    - リソースグループベースのアプリケーション。このオプションを選択すると、このアカウント内でモニターリングするリソースグループを選択できます。コンポーネントで複数のアプリケーションを使用するには、リソースグループベースのモニタリングを使用する必要があります。
    - アカウントベースのアプリケーション。このオプションを選択すると、このアカウント内のすべてのリソースをモニターリングできます。このオプションでは、アプリケーションのオンボーディングプロセスが高速になるため、アカウント内のすべてのリソースをモニターリング

する場合、リソースグループベースのオプションよりもこのオプションを使用することをお勧めします。

> **Note**
>
> Application Insights を使用して、リソースグループベースのモニターリングとアカウントベースのモニターリングを組み合わせることはできません。アプリケーションのタイプを変更するには、モニターリング対象のすべてのアプリケーションを削除して、アプリケーションタイプを選択する必要があります。

モニターリングする最初のアプリケーションを追加すると、CloudWatch Application Insights によってサービスにリンクされたロールがアカウントに作成されます。これにより、お客様に代わって他の AWS サービスを呼び出すアクセス許可が Application Insights に与えられます。Application Insights によってアカウントに作成されるサービスにリンクされたロールについての詳細は、「CloudWatch Application Insights のサービスにリンクされたロールの使用」を参照してください。

3. Resource-based application monitoring

    1. リソースグループを選択します。[Specify application details] (アプリケーションの詳細の指定) ページで、ドロップダウンリストからアプリケーションリソースを含む AWS リソースグループを選択します。これらのリソースには、フロントエンドサーバー、ロードバランサー、Auto Scaling グループ、データベースサーバーが含まれます。

        アプリケーション用のリソースグループをまだ作成していない場合は、[Create new resource group] (新しいリソースグループの作成) を選択して作成できます。リソースグループの作成についての詳細は、AWS Resource Groups ユーザーガイドを参照してください。

    2. CloudWatch Events のモニターリング。CloudWatch Events と Application Insights のモニターリングを統合できるチェックボックスをオンにすると、Amazon EBS、Amazon EC2、AWS CodeDeploy、Amazon ECS、AWS Health APIs And Notifications、Amazon RDS、Amazon S3、および AWS Step Functions からのインサイトを取得できます。

    3. AWS Systems Manager OpsCenter との統合。選択したアプリケーションで問題が検出された場合に通知を表示するには、[Generate Systems Manager OpsCenter OpsItems for remedial actions] (Systems Manager の OpsCenter OpsItems を生成して修復アクションを行う) チェックボックスをオンにします。AWS リソースに関連するオペレーション作

業項目 (OpsItems) を解決するために実行されたオペレーションを追跡するには、SNS トピックの ARN を指定します。

4. タグ – オプション。CloudWatch Application Insights は、タグベースのリソースグループと CloudFormation ベースのリソースグループ (Auto Scaling グループを除く) の両方をサポートしています。詳細については、「タグエディタの使用」を参照してください。

5. [Next] を選択します。

   ARN は、アプリケーション用に次のフォーマットで生成されます。

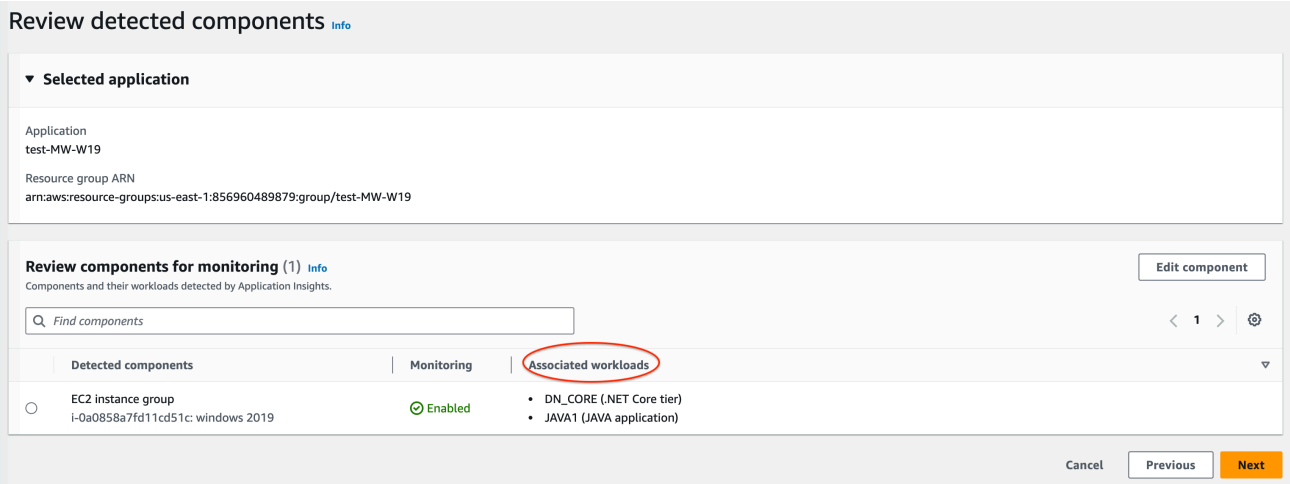
   ```
   arn:partition:applicationinsights:region:account-id:application/resource-group/resource-group-name
   ```

   例

   ```
   arn:aws:applicationinsights:us-east-1:123456789012:application/resource-group/my-resource-group
   ```

6. [検出されたコンポーネントを確認] ページの [モニタリング用のコンポーネントを確認] の下の表には、検出されたコンポーネントとそれに関連して検出されたワークロードが一覧表示されます。

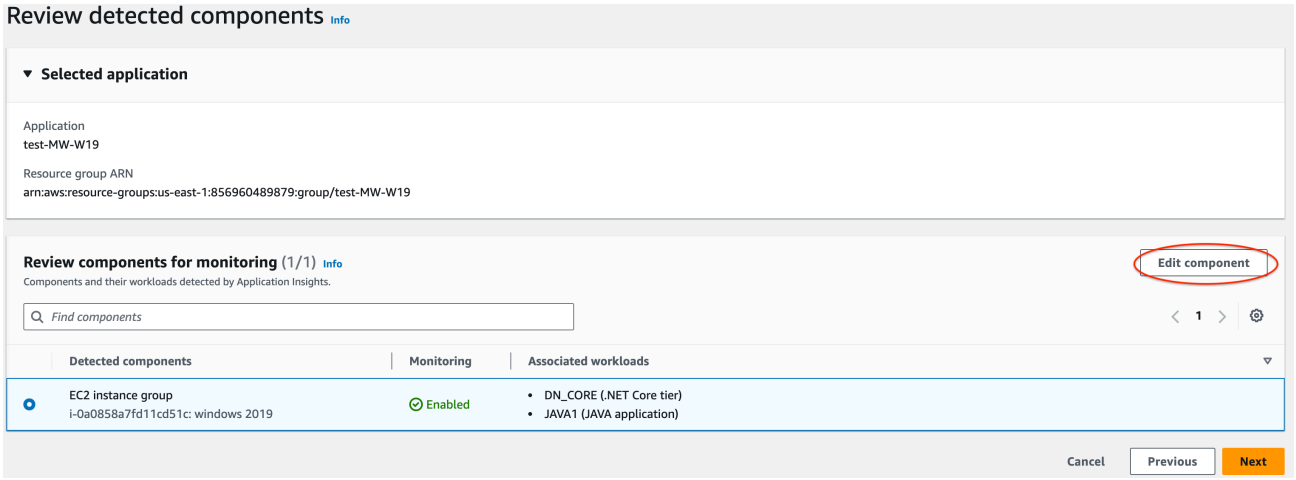
   > **Note**
   >
   > 複数のカスタマイズされたワークロードをサポートするコンポーネントでは、コンポーネントごとに最大 5 つのワークロードをモニタリングできます。これらのワークロードはコンポーネントとは別にモニタリングされます。



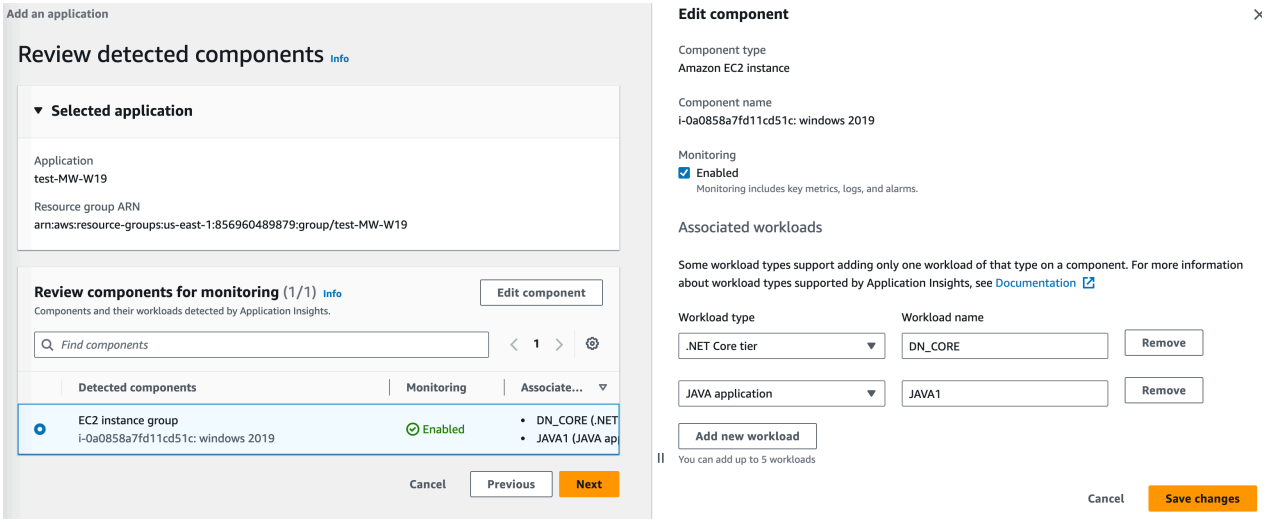
ワークロードがリストされていない場合、[関連付けられたワークロード] にいくつかのメッセージが表示される可能性があります。

- ワークロードを検出できませんでした — ワークロードを検出しようとしたときに問題が発生しました。前提条件 が完了していることを確認してください。ワークロードを追加する必要がある場合は、[コンポーネントを編集] を選択します。
- ワークロードは検出されませんでした — ワークロードは検出されませんでした。ワークロードを追加する必要がある場合があります。そのためには、[コンポーネントを編集] を選択します。
- 該当なし — コンポーネントは、カスタマイズされたワークロードをサポートしていないため、デフォルトのメトリクス、アラーム、ログでモニタリングされます。これらのコンポーネントにはワークロードを追加できません。

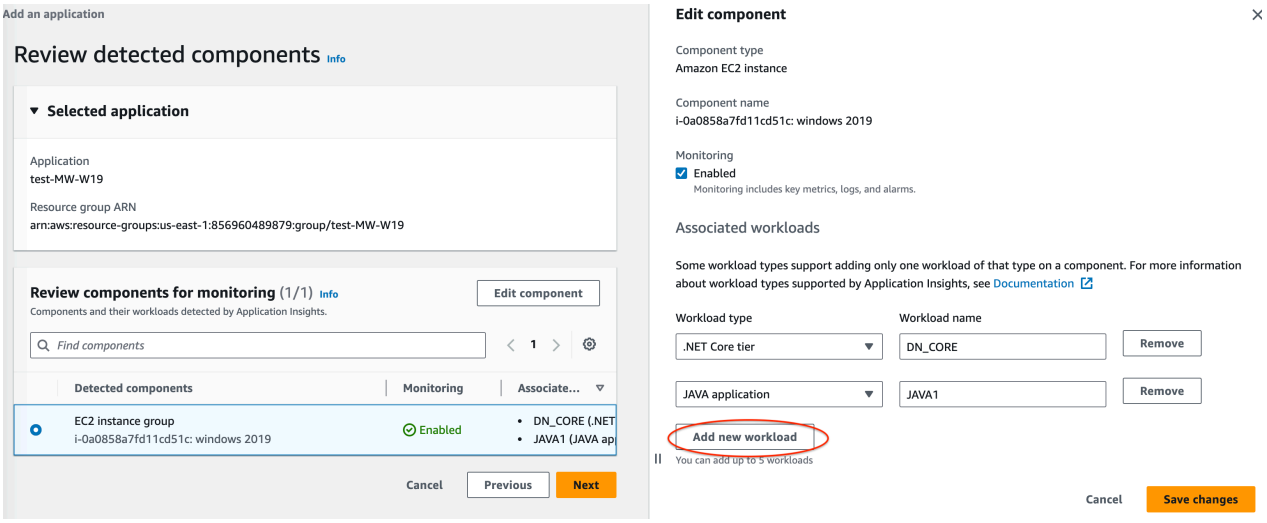
7. コンポーネントを編集するには、コンポーネントを選択し、[コンポーネントを編集] を選択します。サイドパネルが開き、コンポーネントで検出されたワークロードが表示されます。このパネルでは、コンポーネントの詳細を編集したり、新しいワークロードを追加したりできます。



- ワークロードのタイプまたは名前を編集するには、ドロップダウンリストを使用します。

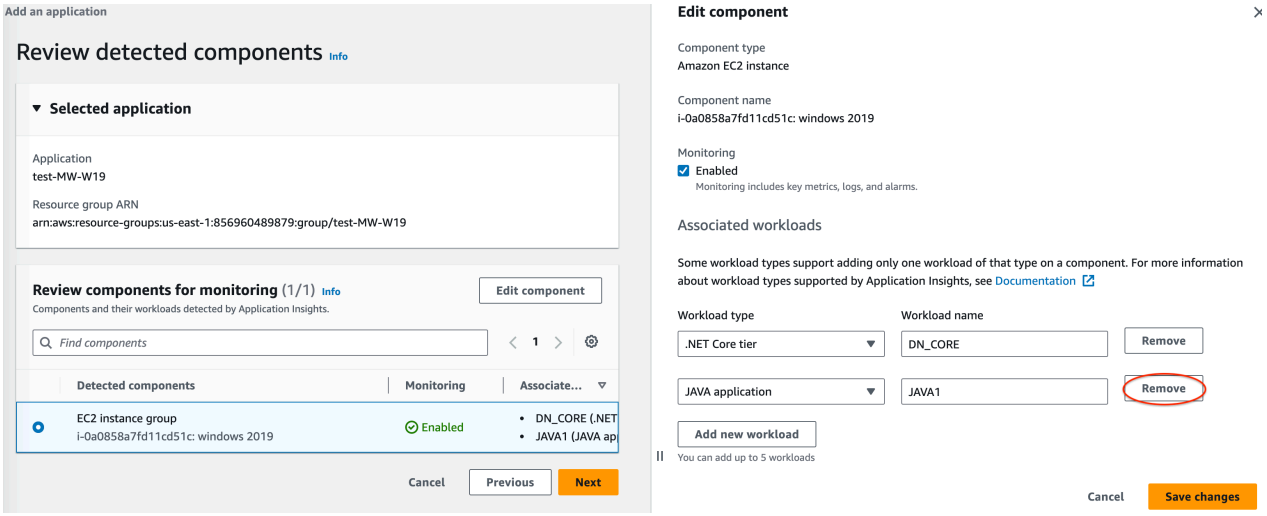


- コンポーネントにワークロードを追加するには、[新しいワークロードを追加] を選択します。

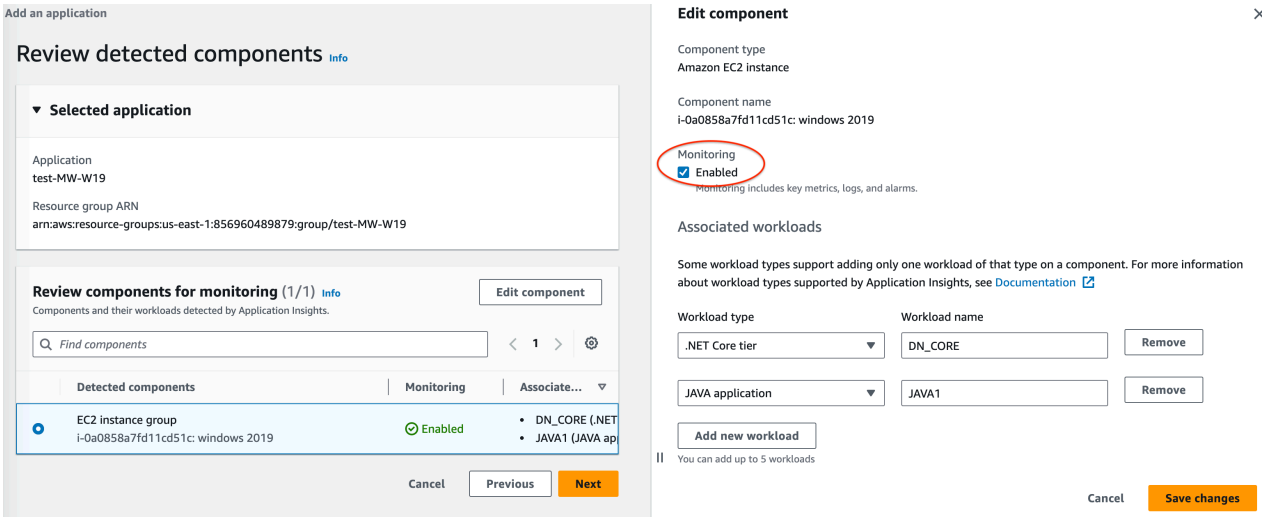


- [新しいワークロードを追加] が表示されない場合は、このコンポーネントは複数のワークロードをサポートしていません。
- 「関連付けられたワークロード」という見出しが表示されない場合、このコンポーネントはカスタマイズされたワークロードをサポートしていません。

- ワークロードを削除するには、モニタリングから削除するワークロードの横にある [削除] を選択します。



- コンポーネント全体のモニタリングを無効にするには、[モニタリング] チェックボックスをオフにします。



- コンポーネントの編集が完了したら、右下隅の [変更を保存] を選択します。コンポーネントのワークロードに加えられた変更は、[関連付けられたワークロード] の下の [モニタリング用のコンポーネントを確認] テーブルに表示されます。

8. [検出されたコンポーネントを確認] ページで [次へ] を選択します。
9. [コンポーネントの詳細を指定] ページには、カスタマイズ可能な関連付けられたワークロードを持つすべてのコンポーネントが含まれています。

> **Note**
>
> コンポーネントヘッダーにオプションのタグが付いている場合、そのコンポーネントのワークロードに関する詳細を追加することはオプションになります。

このページにコンポーネントが表示されない場合、コンポーネントにはこのステップで指定できる追加の詳細情報はありません。

10. [Next] を選択します。
11. [確認して送信] ページで、モニタリング対象のすべてのコンポーネントとワークロードの詳細を確認します。
12. 送信 を選択します。

Account-based application monitoring

1. アプリケーション名。アカウントベースのアプリケーションの名前を入力します。

2. 新しいリソースの自動モニターリング。デフォルトでは、Application Insights では推奨設定が使用され、アプリケーションのオンボーディング後にアカウントに追加されるリソースコンポーネントのモニターリングが設定されます。チェックボックスをオフにすると、アプリケーションのオンボーディング後に追加されるリソースのモニターリングを除外できます。
3. CloudWatch Events のモニターリング。CloudWatch Events と Application Insights のモニターリングを統合できるチェックボックスをオンにすると、Amazon EBS、Amazon EC2、AWS CodeDeploy、Amazon ECS、AWS Health APIs And Notifications、Amazon RDS、Amazon S3、および AWS Step Functions からのインサイトを取得できます。
4. AWS Systems Manager OpsCenter との統合。選択したアプリケーションで問題が検出された場合に通知を表示するには、[Generate Systems Manager OpsCenter OpsItems for remedial actions] (Systems Manager の OpsCenter OpsItems を生成して修復アクションを行う) チェックボックスをオンにします。AWS リソースに関連するオペレーション作業項目 (OpsItems) を解決するために実行されたオペレーションを追跡するには、SNS トピックの ARN を指定します。
5. タグ – オプション。CloudWatch Application Insights は、タグベースのリソースグループと CloudFormation ベースのリソースグループ (Auto Scaling グループを除く) の両方をサポートしています。詳細については、「タグエディタの使用」を参照してください。
6. 検出されるリソース。アカウント内で検出されるすべてのリソースがこのリストに追加されます。Application Insights がアカウント内のリソースをすべて検出できない場合、ページ上部にエラーメッセージが表示されます。このメッセージには、必要なアクセス許可を追加する方法についてのドキュメントへのリンクが含まれます。
7. [Next] を選択します。

   ARN は、アプリケーション用に次のフォーマットで生成されます。

   ```
   arn:partition:applicationinsights:region:account-id:application/
   TBD/application-name
   ```

   例

   ```
   arn:aws:applicationinsights:us-east-1:123456789012:application/TBD/my-
   application
   ```

4. アプリケーションのモニターリング設定を送信すると、アプリケーションの詳細ページが表示され、[Application summary] (アプリケーションの概要)、[Monitored components] (モニターリン

グされているコンポーネント) そして [Unmonitored components] (モニターリングされていないコンポーネント) のリストを確認できます。また、[Components] (コンポーネント) の横にあるタブを選択すると、[Configuration history] (設定履歴)、[Log patterns] (ログのパターン)、適応した [Tags] (タグ) を確認できます。

アプリケーションのインサイトを表示するには、[View Insights] (インサイトを表示する) をクリックします。

[Edit] (編集)を選択すると、CloudWatch Events のモニターリングおよび AWS Systems Manager OpsCenter との統合に関する選択内容を更新できます。

[Components] (コンポーネント) の [Actions] (アクション) メニューを選択して、インスタンスグループを作成、変更、またはグループ解除できます。

コンポーネントの横にある箇条書きから [Manage monitoring] (モニターリングの管理) を選択して、アプリケーション層、ロググループ、イベントログ、メトリクス、カスタムアラームなどのコンポーネントのモニターリングを管理できます。

Amazon ECS および Amazon EKS リソースのモニターリングで Application Insights を有効にする

Application Insights を有効にして、コンテナ化されたアプリケーションとマイクロサービスを Container Insights コンソールからモニターリングできます。Application Insights は、次のリソースのモニターリングをサポートしています。

- Amazon ECS クラスター
- Amazon ECS サービス
- Amazon ECS タスク
- Amazon EKS クラスター

Application Insights を有効にすると、推奨されるメトリクスとログが提供され、潜在的な問題が検出され、CloudWatch Events が生成されます。また、コンテナ化されたアプリケーションとマイクロサービスの自動ダッシュボードが作成されます。

コンテナ化されたリソースの Application Insights は、Container Insights コンソールまたは Application Insights コンソールから有効にできます。

Container Insights コンソールから Application Insights を有効にする

Container Insights コンソール、Container Insights の [Performance monitoring] (パフォーマンスのモニターリング) ダッシュボードで、[Auto-configure Application Insights] (アプリケーションインサイトの自動設定) を選択します。Application Insights を有効にすると、検出された問題の詳細が表示されます。

Application Insights コンソールから Application Insights を有効にする

ECS クラスターがコンポーネントリストに表示されると、Application Insights では Container Insights によるコンテナのモニターリングが追加で自動的に有効になります。

EKS クラスターでは、Container Insights によるモニターリングを追加で有効にできます。これによりコンテナにおける再起動の失敗などの診断情報が提供され、問題の特定と解決に役立てることができます。EKS 用に Container Insights を設定するには、追加の手順が必要です。詳細については「Amazon EKS と Kubernetes での Container Insights のセットアップ」を参照して、EKS で Container Insights を設定する手順をご覧ください。

Container Insights を使用した EKS での追加のモニターリングは、EKS を持つ Linux インスタンスでサポートされています。

ECS および EKS クラスター向けの Container Insights のサポートについての詳細は、「Container Insights」を参照してください。

アプリケーションコンポーネントのモニターリングの無効化

アプリケーションコンポーネントのモニターリングを無効にするには、アプリケーションの詳細ページから、モニターリングを無効にするコンポーネントを選択します。[Actions] (アクション)、[Remove from monitoring] (モニターリングから削除) の順に選択します。

アプリケーションの削除

アプリケーションを削除するには、CloudWatch ダッシュボードの左側のナビゲーションペインで、[Insights] の下にある [Application Insights] を選択します。削除するアプリケーションを選択します。[Actions] (アクション) から、[Delete application] (アプリケーションの削除) を選択します。これにより、モニターリングが削除され、アプリケーションコンポーネントのすべての保存済みモニターリングが削除されます。アプリケーションリソースは削除されません。

コマンドラインを使用してモニターリングするアプリケーションをセットアップ、設定、管理する

このセクションでは、AWS CLI および AWS Tools for Windows PowerShell を使用して、モニターリング用アプリケーションをセットアップ、設定、管理する手順について説明します。

コマンドラインを使用した手順

- アプリケーションの追加と管理
- モニターリングの管理と更新
- SQL Always On 可用性グループのモニターリングを設定する
- MySQL RDS のモニターリングを設定する
- MySQL EC2 のモニターリングを設定する
- PostgreSQL RDS のモニターリングを設定する
- PostgreSQL EC2 のモニターリングを設定する
- Oracle RDS のモニターリングを設定する
- Oracle EC2 のモニターリングを設定する

アプリケーションの追加と管理

コマンドラインを使用して、Application Insights アプリケーションの追加、情報の取得、管理、設定を行うことができます。

トピック

- アプリケーションを追加する
- アプリケーションを記述する
- アプリケーションのコンポーネントをリスト化する
- コンポーネントを記述する
- 類似したリソースをカスタムコンポーネントとしてグループ化する
- カスタムコンポーネントをグループ解除する
- アプリケーションを更新する
- カスタムコンポーネントを更新する

アプリケーションを追加する

AWS CLI を使用してアプリケーションを追加する

AWS CLI を使用して、リソースグループ名が `my-resource-group` のアプリケーションを追加し、作成された opsItem が SNS トピック (ARN `arn:aws:sns:us-`

east-1:123456789012:MyTopic) に対し配信されるように OpsCenter を有効化するには、次のコマンドを使用します。

```
aws application-insights create-application --resource-group-name my-resource-group --
ops-center-enabled --ops-item-sns-topic-arn arn:aws:sns:us-east-1:123456789012:MyTopic
```

AWS Tools for Windows PowerShell を使用してアプリケーションを追加する

AWS Tools for Windows PowerShell を使用して、リソースグループ名が my-resource-group のアプリケーションを追加し、作成された opsItem が SNS トピック ARN arn:aws:sns:us-east-1:123456789012:MyTopic に配信されるように OpsCenter を有効化するには、次のコマンドを使用します。

```
New-CWAIApplication -ResourceGroupName my-resource-group -OpsCenterEnabled true -
OpsItemSNSTopicArn arn:aws:sns:us-east-1:123456789012:MyTopic
```

アプリケーションを記述する

AWS CLI を使用してアプリケーションを記述する

AWS CLI を使用して、my-resource-group というリソースグループに作成されたアプリケーションを記述するには、次のコマンドを使用します。

```
aws application-insights describe-application --resource-group-name my-resource-group
```

AWS Tools for Windows PowerShell を使用してアプリケーションを記述する

AWS Tools for Windows PowerShell を使用して、my-resource-group というリソースグループに作成されたアプリケーションを記述するには、次のコマンドを使用します。

```
Get-CWAIApplication -ResourceGroupName my-resource-group
```

アプリケーションのコンポーネントをリスト化する

AWS CLI を使用してアプリケーションのコンポーネントを一覧表示する

AWS CLI を使用して、my-resource-group というリソースグループで作成されたコンポーネントを一覧表示するには、次のコマンドを使用します。

```
aws application-insights list-components --resource-group-name my-resource-group
```

AWS Tools for Windows PowerShell を使用してアプリケーションのコンポーネントを一覧表示する

AWS Tools for Windows PowerShell を使用して、my-resource-group というリソースグループで作成されたコンポーネントを一覧表示するには、次のコマンドを使用します。

```
Get-CWAIComponentList -ResourceGroupName my-resource-group
```

コンポーネントを記述する

AWS CLI を使用してコンポーネントを記述する

次の AWS CLI コマンドを使用して、my-resource-group というリソースグループで作成されたアプリケーションに属する my-component というコンポーネントを記述できます。

```
aws application-insights describe-component --resource-group-name my-resource-group --
component-name my-component
```

AWS Tools for Windows PowerShell を使用してコンポーネントを記述する

次の AWS Tools for Windows PowerShell コマンドを使用して、my-resource-group というリソースグループで作成されたアプリケーションに属する my-component というコンポーネントを記述できます。

```
Get-CWAIComponent -ComponentName my-component -ResourceGroupName my-resource-group
```

類似したリソースをカスタムコンポーネントとしてグループ化する

類似したリソース (.NET ウェブサーバーインスタンスなど) をカスタムコンポーネントとしてグループ化することをお勧めします。これにより、オンボードが容易になり、モニターリングとインサイトが向上します。現在、CloudWatch Application Insights は EC2 インスタンスのカスタムグループをサポートしています。

AWS CLI を使用してリソースをカスタムコンポーネントとしてグループ化するには

AWS CLI を使用して 3 つのインスタンス (arn:aws:ec2:us-east-1:123456789012:instance/i-11111、arn:aws:ec2:us-

east-1:123456789012:instance/i-22222 および arn:aws:ec2:us-east-1:123456789012:instance/i-33333) をグループ化し、リソースグループ名 my-component で作成されたアプリケーション用としてカスタムコンポーネント my-resource-group に含めるには、次のコマンドを使用します。

```
aws application-insights create-component --resource-group-name my-
resource-group --component-name my-component --resource-list arn:aws:ec2:us-
east-1:123456789012:instance/i-11111 arn:aws:ec2:us-east-1:123456789012:instance/
i-22222 arn:aws:ec2:us-east-1:123456789012:instance/i-33333
```

AWS Tools for Windows PowerShell を使用してリソースをカスタムコンポーネントとしてグループ化するには

AWS Tools for Windows PowerShell を使用して 3 つのインスタンス (arn:aws:ec2:us-east-1:123456789012:instance/i-11111、arn:aws:ec2:us-east-1:123456789012:instance/i-22222、および arn:aws:ec2:us-east-1:123456789012:instance/i-33333) をまとめて、my-component というリソースグループ用に作成されたアプリケーションに対して my-resource-group というカスタムコンポーネントとしてグループ化するには、次のコマンドを使用します。

```
New-CWAIComponent -ResourceGroupName my-resource-group -ComponentName my-component
 -ResourceList arn:aws:ec2:us-east-1:123456789012:instance/i-11111,arn:aws:ec2:us-
east-1:123456789012:instance/i-22222,arn:aws:ec2:us-east-1:123456789012:instance/
i-33333
```

カスタムコンポーネントをグループ解除する

AWS CLI を使用してカスタムコンポーネントのグループ化を解除するには

AWS CLI を使用して、リソースグループ my-resource-group で作成したアプリケーションの my-component という名前のカスタムコンポーネントのグループを解除するには、次のコマンドを使用します。

```
aws application-insights delete-component --resource-group-name my-resource-group --
component-name my-new-component
```

AWS Tools for Windows PowerShell を使用してカスタムコンポーネントのグループ化を解除するには

AWS Tools for Windows PowerShell を使用して、リソースグループ my-resource-group で作成したアプリケーションの my-component という名前のカスタムコンポーネントのグループを解除するには、次のコマンドを使用します。

```
Remove-CWAIComponent -ComponentName my-component -ResourceGroupName my-resource-group
```

アプリケーションを更新する

AWS CLI を使用してアプリケーションを更新する

AWS CLI を使用してアプリケーションを更新し、そのアプリケーションで検出された問題について AWS Systems Manager OpsCenter の OpsItems を生成し、作成された OpsItems を SNS トピック arn:aws:sns:us-east-1:123456789012:MyTopic に関連付けるには、次のコマンドを使用します。

```
aws application-insights update-application --resource-group-name my-resource-group --
ops-center-enabled --ops-item-sns-topic-arn arn:aws:sns:us-east-1:123456789012:MyTopic
```

AWS Tools for Windows PowerShell を使用してアプリケーションを更新する

AWS Tools for Windows PowerShell を使用してアプリケーションを更新し、そのアプリケーションで検出された問題について AWS SSM OpsCenter の OpsItems を生成し、作成された OpsItems を SNS トピック arn:aws:sns:us-east-1:123456789012:MyTopic に関連付けるには、次のコマンドを使用します。

```
Update-CWAIApplication -ResourceGroupName my-resource-group -OpsCenterEnabled true -
OpsItemSNSTopicArn arn:aws:sns:us-east-1:123456789012:MyTopic
```

カスタムコンポーネントを更新する

AWS CLI を使用してカスタムコンポーネントを更新する

AWS CLI を使用して、my-component という名前のカスタムコンポーネントを、新しいコンポーネント名 my-new-component およびインスタンスの更新されたグループで更新するには、次のコマンドを使用します。

```
aws application-insights update-component --resource-group-name my-resource-
group --component-name my-component --new-component-name my-new-component --
```

```
resource-list arn:aws:ec2:us-east-1:123456789012:instance/i-44444 arn:aws:ec2:us-
east-1:123456789012:instance/i-55555
```

AWS Tools for Windows PowerShell を使用したカスタムコンポーネントを更新する

AWS Tools for Windows PowerShell を使用して、my-component という名前のカスタムコンポーネントを、新しいコンポーネント名 my-new-component およびインスタンスの更新されたグループで更新するには、次のコマンドを使用します。

```
Update-CWAIComponent -ComponentName my-component -NewComponentName my-new-
component -ResourceGroupName my-resource-group -ResourceList arn:aws:ec2:us-
east-1:123456789012:instance/i-44444,arn:aws:ec2:us-east-1:123456789012:instance/
i-55555
```

モニターリングの管理と更新

コマンドラインを使用して Application Insights アプリケーションのモニターリングを管理および更新できます。

トピック

- アプリケーションの問題をリスト化する
- アプリケーションの問題を記述する
- 問題に関連する異常やエラーを記述する
- アプリケーションの異常やエラーを記述する
- コンポーネントのモニターリング設定を記述する
- コンポーネントの推奨されるモニターリング設定を記述する
- コンポーネントのモニターリング設定を更新する
- 指定したリソースグループを Application Insights のモニターリングから削除する

アプリケーションの問題をリスト化する

AWS CLI を使用してアプリケーションに関する問題を一覧表示する

AWS CLI を使用して、my-resource-group というリソースグループで作成されたアプリケーションの Unix エポックから 1,000 ~ 10,000 ミリ秒の間で検出されたアプリケーションの問題をリストするには、次のコマンドを使用します。

```
aws application-insights list-problems --resource-group-name my-resource-group --start-
time 1000 --end-time 10000
```

AWS Tools for Windows PowerShell を使用してアプリケーションに関する問題をリストする

AWS Tools for Windows PowerShell を使用して、my-resource-group というリソースグループで作成されたアプリケーションの Unix エポックから 1,000 ~ 10,000 ミリ秒の間で検出されたアプリケーションの問題をリストするには、次のコマンドを使用します。

```
$startDate = "8/6/2019 3:33:00"
$endDate = "8/6/2019 3:34:00"
Get-CWAIProblemList -ResourceGroupName my-resource-group -StartTime $startDate -
EndTime $endDate
```

アプリケーションの問題を記述する

AWS CLI を使用してアプリケーションの問題を説明する

AWS CLI を使用して問題 ID p-1234567890 の問題を記述するには、次のコマンドを使用します。

```
aws application-insights describe-problem –problem-id p-1234567890
```

AWS Tools for Windows PowerShell を使用してアプリケーションの問題を説明する

AWS Tools for Windows PowerShell を使用して問題 ID p-1234567890 の問題を記述するには、次のコマンドを使用します。

```
Get-CWAIProblem -ProblemId p-1234567890
```

問題に関連する異常やエラーを記述する

AWS CLI を使用して問題に関連する異常またはエラーを記述する

AWS CLI を使用して、問題 ID p-1234567890 を持つ問題に関連する異常またはエラーを記述するには、次のコマンドを使用します。

```
aws application-insights describe-problem-observations --problem-id -1234567890
```

AWS Tools for Windows PowerShell を使用して、問題に関連する異常またはエラーを記述する

AWS Tools for Windows PowerShell を使用して、問題 ID p-1234567890 を持つ問題に関連する異常またはエラーを記述するには、次のコマンドを使用します。

```
Get-CWAIProblemObservation -ProblemId p-1234567890
```

アプリケーションの異常やエラーを記述する

AWS CLI を使用してアプリケーションの異常またはエラーを記述する

AWS CLI を使用して、監視 ID o-1234567890 を持つアプリケーションの異常またはエラーを記述するには、次のコマンドを使用します。

```
aws application-insights describe-observation –observation-id o-1234567890
```

AWS Tools for Windows PowerShell を使用して、アプリケーションの異常またはエラーを記述する

AWS Tools for Windows PowerShell を使用して、監視 ID o-1234567890 を持つアプリケーションの異常またはエラーを記述するには、次のコマンドを使用します。

```
Get-CWAIObservation -ObservationId o-1234567890
```

コンポーネントのモニターリング設定を記述する

AWS CLI を使用してコンポーネントのモニターリング設定を記述する

AWS CLI を使用して、リソースグループ my-resource-group で作成されたアプリケーションの my-component というコンポーネントのモニターリング設定を記述するには、次のコマンドを使用します。

```
aws application-insights describe-component-configuration –resource-group-name my-
resource-group –component-name my-component
```

AWS Tools for Windows PowerShell を使用して、コンポーネントのモニターリング設定を記述する

AWS Tools for Windows PowerShell を使用して、リソースグループ my-resource-group で作成されたアプリケーションの my-component というコンポーネントのモニターリング設定を記述するには、次のコマンドを使用します。

```
Get-CWAIComponentConfiguration -ComponentName my-component -ResourceGroupName my-
resource-group
```

コンポーネント設定の詳細と JSON ファイルの例についての詳細は、「コンポーネント設定の作業」を参照してください。

コンポーネントの推奨されるモニターリング設定を記述する

AWS CLI を使用して、コンポーネントの推奨モニターリング設定について記述する

コンポーネントが .NET ワーカーアプリケーションの一部である場合、AWS CLI を使用して、リソースグループ `my-resource-group` で作成されたアプリケーションの `my-component` というコンポーネントの推奨モニターリング設定を記述するには、次のコマンドを使用します。

```
aws application-insights describe-component-configuration-recommendation --resource-
group-name my-resource-group --component-name my-component --tier DOT_NET_WORKER
```

AWS Tools for Windows PowerShell を使用してコンポーネントの推奨モニターリング設定について記述する

コンポーネントが .NET ワーカーアプリケーションの一部である場合、AWS Tools for Windows PowerShell を使用して、リソースグループ `my-resource-group` で作成されたアプリケーションの `my-component` というコンポーネントの推奨モニターリング設定を記述するには、次のコマンドを使用します。

```
Get-CWAIComponentConfigurationRecommendation -ComponentName my-component -
ResourceGroupName my-resource-group -Tier DOT_NET_WORKER
```

コンポーネント設定の詳細と JSON ファイルの例についての詳細は、「コンポーネント設定の作業」を参照してください。

コンポーネントのモニターリング設定を更新する

AWS CLI を使用してコンポーネントのモニターリング設定を更新する

AWS CLI を使用して、`my-component` というリソースグループで作成されたアプリケーションの `my-resource-group` というコンポーネントを更新するには、次のコマンドを使用します。コマンドには、以下のアクションが含まれます。

1. コンポーネントのモニターリングを有効にします。
2. コンポーネントの階層を `.NET Worker` に設定します。
3. コンポーネントの JSON 設定を更新して、ローカルファイル `configuration.txt` から読み取ります。

```
aws application-insights update-component-configuration --resource-group-name my-
resource-group --component-name my-component --tier DOT_NET_WORKER --monitor --
component-configuration "file://configuration.txt"
```

AWS Tools for Windows PowerShell を使用してコンポーネントのモニターリング設定を更新する

AWS Tools for Windows PowerShell を使用して、my-component というリソースグループで作成されたアプリケーションの my-resource-group というコンポーネントを更新するには、次のコマンドを使用します。コマンドには、以下のアクションが含まれます。

1. コンポーネントのモニターリングを有効にします。
2. コンポーネントの階層を .NET Worker に設定します。
3. コンポーネントの JSON 設定を更新して、ローカルファイル configuration.txt から読み取ります。

```
[string]$config = Get-Content -Path configuration.txt
Update-CWAIComponentConfiguration -ComponentName my-component -ResourceGroupName my-
resource-group -Tier DOT_NET_WORKER -Monitor 1 -ComponentConfiguration $config
```

コンポーネント設定の詳細と JSON ファイルの例についての詳細は、「コンポーネント設定の作業」を参照してください。

指定したリソースグループを Application Insights のモニターリングから削除する

AWS CLI を使用して指定されたリソースグループを Application Insights のモニターリングから削除する

AWS CLI を使用して、my-resource-group というリソースグループで作成されたアプリケーションをモニターリングから削除するには、次のコマンドを使用します。

```
aws application-insights delete-application --resource-group-name my-resource-group
```

AWS Tools for Windows PowerShell を使用して指定されたリソースグループを Application Insights のモニターリングから削除する

AWS Tools for Windows PowerShell を使用して、my-resource-group というリソースグループで作成されたアプリケーションをモニターリングから削除するには、次のコマンドを使用します。

```
Remove-CWAIApplication -ResourceGroupName my-resource-group
```

SQL Always On 可用性グループのモニターリングを設定する

1. SQL HA EC2 インスタンスを使用して、リソースグループのアプリケーションを作成します。

```
aws application-insights create-application --region <REGION> --resource-group-name
  <RESOURCE_GROUP_NAME>
```

2. 新しいアプリケーションコンポーネントを作成して、SQL HA クラスターを表す EC2 インスタンスを定義します。

```
aws application-insights create-component --resource-group-name
 "<RESOURCE_GROUP_NAME>" --component-name SQL_HA_CLUSTER --resource-list
  "arn:aws:ec2:<REGION>:<ACCOUNT_ID>:instance/<CLUSTER_INSTANCE_1_ID>"
 "arn:aws:ec2:<REGION>:<ACCOUNT_ID>:instance/<CLUSTER_INSTANCE_2_ID>
```

3. SQL HA コンポーネントを設定します。

```
aws application-insights  update-component-configuration --resource-group-name
 "<RESOURCE_GROUP_NAME>" --region <REGION> --component-name "SQL_HA_CLUSTER" --
monitor --tier SQL_SERVER_ALWAYSON_AVAILABILITY_GROUP --monitor  --component-
configuration '{
  "subComponents" : [ {
    "subComponentType" : "AWS::EC2::Instance",
    "alarmMetrics" : [ {
      "alarmMetricName" : "CPUUtilization",
      "monitor" : true
    }, {
      "alarmMetricName" : "StatusCheckFailed",
      "monitor" : true
    }, {
      "alarmMetricName" : "Processor % Processor Time",
      "monitor" : true
    }, {
      "alarmMetricName" : "Memory % Committed Bytes In Use",
      "monitor" : true
    }, {
      "alarmMetricName" : "Memory Available Mbytes",
      "monitor" : true
    }, {
      "alarmMetricName" : "Paging File % Usage",
```

```
      "monitor" : true
    }, {
      "alarmMetricName" : "System Processor Queue Length",
      "monitor" : true
    }, {
      "alarmMetricName" : "Network Interface Bytes Total/sec",
      "monitor" : true
    }, {
      "alarmMetricName" : "PhysicalDisk % Disk Time",
      "monitor" : true
    }, {
      "alarmMetricName" : "SQLServer:Buffer Manager Buffer cache hit ratio",
      "monitor" : true
    }, {
      "alarmMetricName" : "SQLServer:Buffer Manager Page life expectancy",
      "monitor" : true
    }, {
      "alarmMetricName" : "SQLServer:General Statistics Processes blocked",
      "monitor" : true
    }, {
      "alarmMetricName" : "SQLServer:General Statistics User Connections",
      "monitor" : true
    }, {
      "alarmMetricName" : "SQLServer:Locks Number of Deadlocks/sec",
      "monitor" : true
    }, {
      "alarmMetricName" : "SQLServer:SQL Statistics Batch Requests/sec",
      "monitor" : true
    }, {
      "alarmMetricName" : "SQLServer:Database Replica File Bytes Received/sec",
      "monitor" : true
    }, {
      "alarmMetricName" : "SQLServer:Database Replica Log Bytes Received/sec",
      "monitor" : true
    }, {
      "alarmMetricName" : "SQLServer:Database Replica Log remaining for undo",
      "monitor" : true
    }, {
      "alarmMetricName" : "SQLServer:Database Replica Log Send Queue",
      "monitor" : true
    }, {
      "alarmMetricName" : "SQLServer:Database Replica Mirrored Write Transaction/
sec",
      "monitor" : true
```

```
    }, {
      "alarmMetricName" : "SQLServer:Database Replica Recovery Queue",
      "monitor" : true
    }, {
      "alarmMetricName" : "SQLServer:Database Replica Redo Bytes Remaining",
      "monitor" : true
    }, {
      "alarmMetricName" : "SQLServer:Database Replica Redone Bytes/sec",
      "monitor" : true
    }, {
      "alarmMetricName" : "SQLServer:Database Replica Total Log requiring undo",
      "monitor" : true
    }, {
      "alarmMetricName" : "SQLServer:Database Replica Transaction Delay",
      "monitor" : true
    } ],
    "windowsEvents" : [ {
      "logGroupName" : "WINDOWS_EVENTS-Application-<RESOURCE_GROUP_NAME>",
      "eventName" : "Application",
      "eventLevels" : [ "WARNING", "ERROR", "CRITICAL", "INFORMATION" ],
      "monitor" : true
    }, {
      "logGroupName" : "WINDOWS_EVENTS-System-<RESOURCE_GROUP_NAME>",
      "eventName" : "System",
      "eventLevels" : [ "WARNING", "ERROR", "CRITICAL" ],
      "monitor" : true
    }, {
      "logGroupName" : "WINDOWS_EVENTS-Security-<RESOURCE_GROUP_NAME>",
      "eventName" : "Security",
      "eventLevels" : [ "WARNING", "ERROR", "CRITICAL" ],
      "monitor" : true
    } ],
    "logs" : [ {
      "logGroupName" : "SQL_SERVER_ALWAYSON_AVAILABILITY_GROUP-
<RESOURCE_GROUP_NAME>",
      "logPath" : "C:\\Program Files\\Microsoft SQL Server\\MSSQL**.MSSQLSERVER\
\MSSQL\\Log\\ERRORLOG",
      "logType" : "SQL_SERVER",
      "monitor" : true,
      "encoding" : "utf-8"
    } ]
  }, {
    "subComponentType" : "AWS::EC2::Volume",
    "alarmMetrics" : [ {
```

```
      "alarmMetricName" : "VolumeReadBytes",
      "monitor" : true
    }, {
    "alarmMetricName" : "VolumeWriteBytes",
      "monitor" : true
    }, {
    "alarmMetricName" : "VolumeReadOps",
      "monitor" : true
    }, {
    "alarmMetricName" : "VolumeWriteOps",
      "monitor" : true
    }, {
    "alarmMetricName" : "VolumeQueueLength",
      "monitor" : true
    }, {
    "alarmMetricName" : "VolumeThroughputPercentage",
      "monitor" : true
    }, {
    "alarmMetricName" : "BurstBalance",
      "monitor" : true
    } ]
  } ]
}'
```

> **Note**
>
> Application Insights は、フェイルオーバーなどのクラスターアクティビティを検出するために、アプリケーションイベントログ (情報レベル) を取り込む必要があります。

MySQL RDS のモニターリングを設定する

1. RDS MySQL データベースインスタンスを使用して、リソースグループのアプリケーションを作成します。

```
aws application-insights create-application --region <REGION> --resource-group-name
  <RESOURCE_GROUP_NAME>
```

2. エラーログはデフォルトで有効になります。スロークエリログは、データパラメータグループを使用して有効にできます。詳細については、「MySQL スロークエリと一般ログにアクセスする」を参照してください。

- `set slow_query_log = 1`
- `set log_output = FILE`

3. モニターリングするログを CloudWatch Logs にエクスポートします。詳細については、「CloudWatch Logs への MySQL ログの発行」を参照してください。
4. MySQL RDS コンポーネントを設定します。

```
aws application-insights  update-component-configuration --resource-group-name
 "<RESOURCE_GROUP_NAME>" --region <REGION> --component-name "<DB_COMPONENT_NAME>"
 --monitor --tier DEFAULT --monitor  --component-configuration "{\"alarmMetrics\":
[{\"alarmMetricName\":\"CPUUtilization\",\"monitor\":true}],\"logs\":[{\"logType\":
\"MYSQL\",\"monitor\":true},{\"logType\": \"MYSQL_SLOW_QUERY\",\"monitor\":false}]}"
```

MySQL EC2 のモニターリングを設定する

1. SQL HA EC2 インスタンスを使用して、リソースグループのアプリケーションを作成します。

```
aws application-insights create-application --region <REGION> --resource-group-name
  <RESOURCE_GROUP_NAME>
```

2. エラーログはデフォルトで有効になります。スロークエリログは、データパラメータグループを使用して有効にできます。詳細については、「MySQL スロークエリと一般ログにアクセスする」を参照してください。
   - `set slow_query_log = 1`
   - `set log_output = FILE`
3. MySQL EC2 コンポーネントを設定します。

```
aws application-insights  update-component-configuration --resource-group-name
 "<RESOURCE_GROUP_NAME>" --region <REGION> --component-name "<DB_COMPONENT_NAME>"
 --monitor --tier MYSQL --monitor  --component-configuration "{\"alarmMetrics\":
[{\"alarmMetricName\":\"CPUUtilization\",\"monitor\":true}],\"logs\":[{\"logGroupName
\":\"<UNIQUE_LOG_GROUP_NAME>\",\"logPath\":\"C:\\\\ProgramData\\\\MySQL\\\\MySQL
 Server **\\\\Data\\\\<FILE_NAME>.err\",\"logType\":\"MYSQL\",\"monitor\":true,
\"encoding\":\"utf-8\"}]}"
```

PostgreSQL RDS のモニターリングを設定する

1. PostgreSQL RDS データベースインスタンスを使用して、リソースグループのアプリケーションを作成します。

```
aws application-insights create-application --region <REGION> --resource-group-name
  <RESOURCE_GROUP_NAME>
```

2. CloudWatch への PostgreSQL ログの公開は、デフォルトでは有効になっていません。モニターリングを有効にするには、RDS コンソールを開き、モニターリングするデータベースを選択します。右上隅にある [Modify] (変更) を選択し、[PostgreSQL] ログというラベルの付いたチェックボックスをオンにします。[Continue (続行)] を選択してこの設定を保存します。
3. PostgreSQL のログが CloudWatch にエクスポートされます。
4. PostgreSQL RDS コンポーネントを設定します。

```
aws application-insights update-component-configuration --region <REGION> --resource-
group-name <RESOURCE_GROUP_NAME> --component-name <DB_COMPONENT_NAME> --monitor --
tier DEFAULT --component-configuration
"{
   \"alarmMetrics\":[
      {
         \"alarmMetricName\": \"CPUUtilization\",
         \"monitor\": true
      }
   ],
   \"logs\":[
      {
         \"logType\": \"POSTGRESQL\",
         \"monitor\": true
      }
   ]
}"
```

PostgreSQL EC2 のモニターリングを設定する

1. PostgreSQL EC2 インスタンスを使用して、リソースグループのアプリケーションを作成します。

```
aws application-insights create-application --region <REGION> --resource-group-name
  <RESOURCE_GROUP_NAME>
```

2. PostgreSQL EC2 コンポーネントを設定します。

```
aws application-insights update-component-configuration --region <REGION> --resource-
group-name <RESOURCE_GROUP_NAME> --component-name <DB_COMPONENT_NAME> --monitor --
tier POSTGRESQL --component-configuration
"{
   \"alarmMetrics\":[
      {
         \"alarmMetricName\":\"CPUUtilization\",
         \"monitor\":true
      }
   ],
   \"logs\":[
      {
         \"logGroupName\":\"<UNIQUE_LOG_GROUP_NAME>\",
         \"logPath\":\"/var/lib/pgsql/data/log/\",
         \"logType\":\"POSTGRESQL\",
         \"monitor\":true,
         \"encoding\":\"utf-8\"
      }
   ]
}"
```

Oracle RDS のモニターリングを設定する

1. Oracle RDS データベースインスタンスを使用して、リソースグループのアプリケーションを作成します。

```
aws application-insights create-application --region <REGION> --resource-group-name
  <RESOURCE_GROUP_NAME>
```

2. CloudWatch への Oracle ログの発行は、デフォルトでは有効になっていません。モニターリングを有効にするには、RDS コンソールを開き、モニターリングするデータベースを選択します。右上隅にある [Modify] (変更) を選択し、[Alert] (アラート) ログと [Listener] (リスナー) ログというラベルの付いたチェックボックスをオンにします。[Continue (続行)] を選択してこの設定を保存します。

3. Oracle ログは CloudWatch にエクスポートされます。

4. Oracle RDS コンポーネントを設定します。

```
aws application-insights update-component-configuration --region <REGION> --resource-
group-name <RESOURCE_GROUP_NAME> --component-name <DB_COMPONENT_NAME> --monitor --
tier DEFAULT --component-configuration
"{
   \"alarmMetrics\":[
      {
         \"alarmMetricName\": \"CPUUtilization\",
         \"monitor\": true
      }
   ],
   \"logs\":[
      {
         \"logType\": \"ORACLE_ALERT\",
         \"monitor\": true
      },
      {
         \"logType\": \"ORACLE_LISTENER\",
         \"monitor\": true
      }
   ]
}"
```

Oracle EC2 のモニターリングを設定する

1. Oracle EC2 インスタンスを使用して、リソースグループのアプリケーションを作成します。

```
aws application-insights create-application --region <REGION> --resource-group-name
  <RESOURCE_GROUP_NAME>
```

2. Oracle EC2 コンポーネントを設定します。

```
aws application-insights update-component-configuration --region <REGION> --resource-
group-name <RESOURCE_GROUP_NAME> --component-name <DB_COMPONENT_NAME> --monitor --
tier ORACLE --component-configuration
"{
   \"alarmMetrics\":[
      {
         \"alarmMetricName\":\"CPUUtilization\",
```

```
         \"monitor\":true
      }
   ],
   \"logs\":[
      {
         \"logGroupName\":\"<UNIQUE_LOG_GROUP_NAME>\",
         \"logPath\":\"/opt/oracle/diag/rdbms/*/*/trace\",
         \"logType\":\"ORACLE_ALERT\",
         \"monitor\":true,
      },
      {
         \"logGroupName\":\"<UNIQUE_LOG_GROUP_NAME>\",
         \"logPath\":\"/opt/oracle/diag/tnslsnr/$HOSTNAME/listener/trace/\",
         \"logType\":\"ORACLE_ALERT\",
         \"monitor\":true,
      }
   ]
}"
```

Application Insights CloudWatch Events と検出された問題の通知

CloudWatch Application Insights に追加したアプリケーションごとに、CloudWatch イベントがベストエフォートベースで以下のイベントに対して発行されます。

- 問題の作成。CloudWatch Application Insights が新しい問題を検出したときに発行されます。
  - 詳細タイプ: 「Application Insights 問題が検出されました」
  - 詳細:
    - `problemId`: 検出された問題 ID。
    - `region`: 問題が作成された AWS リージョン。
    - `resourceGroupName`: 問題が検出された登録済みアプリケーションのリソースグループ。
    - `status`: 問題のステータス。指定できるステータスと定義は次のとおりです。
      - `In progress`: 新しい問題が確認されました。問題に関する監視結果を受信中です。
      - `Recovering`: この問題は安定化されています。この状態になると、手動で問題を解決できます。
      - `Resolved`: この問題は解決されました。この問題に関する新たな監視結果はありません。
      - `Recurring`: この問題は過去 24 時間以内に解決されました。追加監視の結果、再開しました。

    - `severity`: 問題の重大度。
    - `problemUrl`: 問題のコンソール URL。

- 問題の更新。問題が新しい監視結果で更新されたとき、または既存の監視結果が更新されて問題が後で更新されたときに発行されます。更新には、問題の解決や終了が含まれます。
    - 詳細タイプ: 「Application Insights の問題が更新されました」
    - 詳細:
        - `problemId`: 作成された問題 ID
        - `region`: 問題が作成された AWS リージョン。
        - `resourceGroupName`: 問題が検出された登録済みアプリケーションのリソースグループ。
        - `status`: 問題のステータス。
        - `severity`: 問題の重大度。
        - `problemUrl`: 問題のコンソール URL。

アプリケーションによって生成された問題イベントの通知を受信する方法

CloudWatch コンソールの左側のナビゲーションペインで、[イベント] の下の [ルール] を選択します。[ルール] ページで、[ルールの作成] を選択します。[サービス名] ドロップダウンリストから [Amazon CloudWatch Application Insights] を選択し、[イベントタイプ] を選択します。次に、[ターゲットの追加] を選択し、ターゲットとパラメータ (SNS トピックや Lambda 関数など) を選択します。

AWS Systems Manager を通じたアクション。CloudWatch Application Insights には、Systems Manager OpsCenter との統合が組み込まれています。アプリケーションにこの統合を使用する場合、アプリケーションで検出された問題ごとに OpsCenter コンソールに OpsItem が作成されます。OpsCenter コンソールから、CloudWatch Application Insights で検出された問題に関する要約情報を表示し、Systems Manager Automation ランブックを選択して修復アクションを実行したり、アプリケーションでリソースの問題を引き起こしている Windows プロセスをさらに特定したりできます。

アプリケーションインサイトのクロスアカウントオブザーバビリティ

CloudWatch Application Insights のクロスアカウントオブザーバビリティを使用すると、単一リージョン内の複数の AWS アカウントにまたがるアプリケーションをモニタリングし、トラブルシューティングすることができます。

Amazon CloudWatch オブザーバビリティアクセスマネージャーを使用して、1 つ以上の AWS アカウントをモニタリングアカウントとして設定できます。モニタリングアカウントにシンクを作成することで、モニタリングアカウントにソースアカウントのデータを表示する機能を持たせます。シンクを使用して、ソースアカウントからモニタリングアカウントへのリンクを作成します。詳細については、「CloudWatch のクロスアカウントオブザーバビリティ」を参照してください。

必要なリソース

CloudWatch Application Insights のクロスアカウントオブザーバビリティが適切に機能するためには、以下のテレメトリータイプが CloudWatch Observability Access Manager を通じて共有されていることを確認してください。

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

コンポーネント設定の作業

コンポーネント設定は、コンポーネントの設定を記述する JSON 形式のテキストファイルです。このセクションでは、テンプレートフラグメントの例、コンポーネント設定セクションの説明、コンポーネント設定例を示します。

トピック

- コンポーネント設定テンプレートのフラグメント
- コンポーネント設定セクション
- コンポーネント設定の例

コンポーネント設定テンプレートのフラグメント

次の例では、JSON 形式のテンプレートのフラグメントを示しています。

```
{
  "alarmMetrics" : [
    list of alarm metrics
  ],
  "logs" : [
    list of logs
```

```
  ],
  "processes" : [
   list of processes
  ],
  "windowsEvents" : [
    list of windows events channels configurations
  ],
  "alarms" : [
    list of CloudWatch alarms
  ],
  "jmxPrometheusExporter": {
    JMX Prometheus Exporter configuration
  },
  "hanaPrometheusExporter": {
      SAP HANA Prometheus Exporter configuration
  },
  "haClusterPrometheusExporter": {
      HA Cluster Prometheus Exporter configuration
  },
  "netWeaverPrometheusExporter": {
      SAP NetWeaver Prometheus Exporter configuration
  },
  "subComponents" : [
    {
      "subComponentType" : "AWS::EC2::Instance" ...
      component nested instances configuration
    },
    {
      "subComponentType" : "AWS::EC2::Volume" ...
      component nested volumes configuration
    }
  ]
}
```

コンポーネント設定セクション

コンポーネントの設定には、いくつかの主要なセクションがあります。コンポーネント設定のセクションは、任意の順序でリストできます。

- alarmMetrics (オプション)

  コンポーネントでモニターリングするメトリクスのリスト。すべてのコンポーネントタイプに alarmMetrics セクションを設定できます。

- logs (オプション)

  コンポーネントでモニターリングするログのリスト。logs セクションは EC2 インスタンスのみに設定できます。

- プロセス (オプション)

  コンポーネントでモニターリングするプロセスのリスト。プロセスセクションは EC2 インスタンスのみに設定できます。

- subComponents (オプション)

  コンポーネントのネストされたインスタンスおよびボリューム subComponent 設定。次のタイプのコンポーネントには、ネストされたインスタンスと subComponents セクションを含めることができます。ELB、ASG、カスタムグループ化された EC2 インスタンス、および EC2 インスタンス。

- アラーム (オプション)

  コンポーネントでモニターリングするアラームのリスト。すべてのコンポーネントタイプにアラームセクションを設定できます。

- windowsEvents (オプション)

  コンポーネントでモニターリングする Windows イベントのリスト。EC2 インスタンスの Windows にのみ `windowsEvents` セクションがあります。

- JMXPrometheusExporter (オプション)

  JMXPrometheus Exporter の設定。

- hanaPrometheusExporter (オプション)

  SAP HANA Prometheus Exporter の設定。

- haClusterPrometheusExporter (オプション)

  HA Cluster Prometheus Exporter の設定。

- netWeaverPrometheusExporter (オプション)

  SAP NetWeaver Prometheus Exporter の設定。

- sapAsePrometheusExporter (オプション)

  SAP ASE Prometheus Exporter の設定。

次の例は、JSON 形式の subComponents セクションフラグメントの構文を示しています。

```
[
  {
    "subComponentType" : "AWS::EC2::Instance",
    "alarmMetrics" : [
      list of alarm metrics
    ],
    "logs" : [
      list of logs
    ],
    "processes": [
      list of processes
    ],
    "windowsEvents" : [
      list of windows events channels configurations
    ]
  },
  {
    "subComponentType" : "AWS::EC2::Volume",
    "alarmMetrics" : [
      list of alarm metrics
    ]
  }
]
```

コンポーネント設定セクションのプロパティ

このセクションでは、各コンポーネント設定セクションのプロパティについて説明します。

セクション

- メトリクス
- Log
- プロセス
- JMX Prometheus Exporter
- HANA Prometheus Exporter
- HA Cluster Prometheus Exporter
- NetWeaver Prometheus Exporter
- SAP ASE Prometheus Exporter

- Windows イベント
- アラーム

メトリクス

コンポーネントでモニターリングするメトリクスを定義します。

JSON

```
{
  "alarmMetricName" : "monitoredMetricName",
  "monitor" : true/false
}
```

プロパティ

- alarmMetricName (必須)

  コンポーネントでモニターリングされるメトリクスの名前。Application Insights でサポートされるメトリクスについては、「Amazon CloudWatch Application Insights でサポートされるログとメトリクス」を参照してください。
- monitor (オプション)

  メトリクスをモニターリングするかどうかを示すブール値。デフォルト値は `true` です。

Log

コンポーネントでモニターリングするログを定義します。

JSON

```
{
  "logGroupName" : "logGroupName",
  "logPath" : "logPath",
  "logType" : "logType",
  "encoding" : "encodingType",
  "monitor" : true/false
}
```

プロパティ

- logGroupName (必須)

  モニターリングされたログに関連付けられる CloudWatch Logs グループ名。ロググループ名の制約については、「CreateLogGroup」を参照してください。

- logPath (EC2 インスタンスのコンポーネントでは必須、AWS Lambda などの CloudWatch エージェントを使用しないコンポーネントでは不要)

  モニターリングするログのパス。ログパスは、Windows システムファイルの絶対パスである必要があります。詳細については、「CloudWatch エージェント設定ファイル: ログセクション」を参照してください。

- logType (必須)

  ログタイプは、Application Insights がログを分析するログパターンを決定します。ログの種類は以下から選択します。

  - SQL_SERVER
  - MYSQL
  - MYSQL_SLOW_QUERY
  - POSTGRESQL
  - ORACLE_ALERT
  - ORACLE_LISTENER
  - IIS
  - APPLICATION
  - WINDOWS_EVENTS
  - WINDOWS_EVENTS_ACTIVE_DIRECTORY
  - WINDOWS_EVENTS_DNS
  - WINDOWS_EVENTS_IIS
  - WINDOWS_EVENTS_SHAREPOINT
  - SQL_SERVER_ALWAYSON_AVAILABILITY_GROUP
  - SQL_SERVER_FAILOVER_CLUSTER_INSTANCE
  - DEFAULT
  - CUSTOM
  - STEP_FUNCTION
  - API_GATEWAY_ACCESS

  - API_GATEWAY_EXECUTION
  - SAP_HANA_LOGS
  - SAP_HANA_TRACE
  - SAP_HANA_HIGH_AVAILABILITY
  - SAP_NETWEAVER_DEV_TRACE_LOGS
  - PACEMAKER_HIGH_AVAILABILITY
- encoding (オプション)

  モニターリングするログのエンコードのタイプ。指定するエンコードは、CloudWatch エージェントでサポートされているエンコードのリストに含まれている必要があります。指定しない場合、CloudWatch Application Insights は、以下を除き、デフォルトのエンコーディング タイプ utf-8 を使用します。
  - SQL_SERVER: UTF-16 エンコード
  - IIS: ASCII エンコード
- monitor (オプション)

  ログをモニターリングするかどうかを示すブール値。デフォルト値は true です。

プロセス

コンポーネントでモニターリングするプロセスを定義します。

JSON

```
{
  "processName" : "monitoredProcessName",
  "alarmMetrics" : [
      list of alarm metrics
  ]
}
```

プロパティ

- processName (必須)

  コンポーネントでモニターリングされるプロセスの名前。プロセスの名前には、sqlservr または sqlservr.exe のようなプロセスステムを含めることはできません。

- alarmMetric (必須)

  このプロセスをモニターするメトリクスのリスト。CloudWatch Application Insights でサポートされるプロセスメトリクスを表示するには、「 Amazon Elastic Compute Cloud (EC2) 」を参照してください。

JMX Prometheus Exporter

JMX Prometheus Exporter の設定を定義します。

JSON

```
"JMXPrometheusExporter": {
  "jmxURL" : "JMX URL",
  "hostPort" : "The host and port",
  "prometheusPort" : "Target port to emit Prometheus metrics"
}
```

プロパティ

- jmxURL (オプション)

  接続する完全な JMX URL。

- hostPort (オプション)

  リモート JMX 経由で接続するホストとポート。jmxURL と hostPort のいずれかのみを指定できます。

- prometheusPort (オプション)

  Prometheus メトリクスを送信するターゲットポート。指定しない場合は、デフォルトのポート 9404 が使用されます。

HANA Prometheus Exporter

HANA Prometheus Exporter の設定を定義します。

JSON

```
"hanaPrometheusExporter": {
```

```
    "hanaSid": "SAP HANA  SID",
    "hanaPort": "HANA database port",
    "hanaSecretName": "HANA secret name",
    "prometheusPort": "Target port to emit Prometheus metrics"
}
```

プロパティ

- hanaSid

  SAP HANA システムの 3 文字の SAP システム ID (SID)。
- hanaPort

  エクスポーターが HANA メトリクスをクエリするために使用する HANA データベースのポート。
- hanaSecretName

  HANA のモニターリングにおけるユーザー認証情報を格納する AWS Secrets Manager シークレット。HANA Prometheus Exporter では、これらの認証情報を使用してデータベースに接続し、HANA メトリクスをクエリします。
- prometheusPort (オプション)

  Prometheus によるメトリクスの送信先となるターゲットポート。指定しない場合は、デフォルトのポート 9668 が使用されます。

HA Cluster Prometheus Exporter

HA Cluster Prometheus Exporter の設定を定義します。

JSON

```
"haClusterPrometheusExporter": {
    "prometheusPort": "Target port to emit Prometheus metrics"
}
```

プロパティ

- prometheusPort (オプション)

  Prometheus によるメトリクスの送信先となるターゲットポート。指定しない場合は、デフォルトのポート 9664 が使用されます。

NetWeaver Prometheus Exporter

NetWeaver Prometheus Exporter の設定を定義します。

JSON

```
"netWeaverPrometheusExporter": {
    "sapSid": "SAP NetWeaver  SID",
    "instanceNumbers": [ "Array of instance Numbers of SAP NetWeaver system "],
"prometheusPort": "Target port to emit Prometheus metrics"
}
```

プロパティ

- sapSid

  SAP Netweaver システムの 3 文字の SAP システム ID (SID)。
- instanceNumbers

  SAP NetWeaver システムのインスタンス番号の配列。

  例: `"instanceNumbers": [ "00", "01"]`
- prometheusPort (オプション)

  Prometheus メトリクスの送信先となるターゲットポート。指定しない場合、デフォルトのポート 9680 が使用されます。

SAP ASE Prometheus Exporter

SAP ASE Prometheus Exporter の設定を定義します。

JSON

```
"sapASEPrometheusExporter": {
    "sapAseSid": "SAP ASE SID",
    "sapAsePort": "SAP ASE database port",
    "sapAseSecretName": "SAP ASE secret name",
    "prometheusPort": "Target port to emit Prometheus metrics",
    "agreeToEnableASEMonitoring": true
}
```

プロパティ

- sapAseSid

  SAP ASE システムの 3 文字の SAP システム ID (SID)。
- sapAsePort

  エクスポーターが ASE メトリクスをクエリするために使用する SAP ASE データベースのポート。
- sapAseSecretName

  ASE のモニタリングにおけるユーザー認証情報を格納する AWS Secrets Manager シークレット。SAP ASE Prometheus Exporter では、これらの認証情報を使用してデータベースに接続し、ASE メトリクスをクエリします。
- prometheusPort (オプション)

  Prometheus によるメトリクスの送信先となるターゲットポート。指定しない場合は、デフォルトのポート 9399 が使用されます。デフォルトのポートを使用している別の ASE DB がある場合は、9499 を使用します。

Windows イベント

ログに記録する Windows イベントを定義します。

JSON

```
{
  "logGroupName" : "logGroupName",
  "eventName" : "eventName",
  "eventLevels" : ["ERROR","WARNING","CRITICAL","INFORMATION","VERBOSE"],
  "monitor" : true/false
}
```

プロパティ

- logGroupName (必須)

  モニターリングされたログに関連付けられる CloudWatch Logs グループ名。ロググループ名の制約については、「CreateLogGroup」を参照してください。
- eventName (必須)

ログ記録する Windows イベントのタイプ。これは Windows イベントログのチャネル名と同等です。たとえば、System、Security、CustomEventName などです。このフィールドは、ログ記録する Windows イベントのタイプごとに必要です。

- eventLevels (必須)

  ログ記録するイベントのレベル。ログ記録する各レベルを指定する必要があります。指定できる値には、INFORMATION、WARNING、ERROR、CRITICALおよび VERBOSE などがあります。このフィールドは、ログ記録する Windows イベントのタイプごとに必要です。

- monitor (オプション)

  ログをモニターリングするかどうかを示すブール値。デフォルト値は `true` です。

アラーム

コンポーネントでモニターリングする CloudWatch アラームを定義します。

JSON

```
{
  "alarmName" : "monitoredAlarmName",
  "severity" : HIGH/MEDIUM/LOW
}
```

プロパティ

- alarmName (必須)

  コンポーネントでモニターリングする CloudWatch アラームの名前。

- 重大度 (オプション)

  アラームが鳴ったときの停止の程度を示します。

コンポーネント設定の例

以下の例は、関連するサービスの JSON 形式のコンポーネント設定を示しています。

コンポーネント設定の例

- Amazon DynamoDB テーブル

- Amazon EC2 Auto Scaling (ASG)
- Amazon EKS クラスター
- Amazon Elastic Compute Cloud (EC2) インスタンス
- Amazon Elastic Container Service (Amazon ECS)
- Amazon ECS サービス
- Amazon ECS タスク
- Amazon Elastic File System (Amazon EFS)
- Amazon FSx
- Amazon Relational Database Service (RDS) Aurora MySQL
- Amazon Relational Database Service (RDS) インスタンス
- Amazon Route 53 ヘルスチェック
- Amazon Route 53 ホストゾーン
- Amazon Route 53 Resolver エンドポイント
- Amazon Route 53 Resolver クエリログ記録の設定
- Amazon S3 バケット
- Amazon Simple Queue Service (SQS)
- Amazon SNS トピック
- Amazon Virtual Private Cloud (Amazon VPC)
- Amazon VPC ネットワークアドレス変換 (NAT) ゲートウェイ
- API Gateway REST API ステージ
- Application Elastic Load Balancing
- AWS Lambda 関数
- AWS Network Firewall ルールグループ
- AWS Network Firewall ルールグループ関連付け
- AWS Step Functions
- カスタマーグループ化された Amazon EC2 インスタンス
- Elastic Load Balancing
- Java
- Amazon EC2 での Kubernetes
- RDS MariaDB と RDS MySQL

- RDS Oracle
- RDS PostgreSQL
- Amazon EC2 での SAP ASE
- Amazon EC2 での SAP ASE High Availability
- Amazon EC2 での SAP HANA
- Amazon EC2 での SAP HANA High Availability
- Amazon EC2 での SAP NetWeaver
- Amazon EC2 での SAP Netweaver High Availability
- SQL Always On 可用性グループ
- SQL フェイルオーバークラスターインスタンス

Amazon DynamoDB テーブル

次の例は、Amazon DynamoDB テーブル用の JSON 形式のコンポーネント設定を示しています。

```
{
  "alarmMetrics": [
    {
      "alarmMetricName": "SystemErrors",
      "monitor": false
    },
    {
      "alarmMetricName": "UserErrors",
      "monitor": false
    },
    {
      "alarmMetricName": "ConsumedReadCapacityUnits",
      "monitor": false
    },
    {
      "alarmMetricName": "ConsumedWriteCapacityUnits",
      "monitor": false
    },
    {
      "alarmMetricName": "ReadThrottleEvents",
      "monitor": false
    },
    {
      "alarmMetricName": "WriteThrottleEvents",
```

```
      "monitor": false
    },
    {
      "alarmMetricName": "ConditionalCheckFailedRequests",
      "monitor": false
    },
    {
      "alarmMetricName": "TransactionConflict",
      "monitor": false
    }
  ],
  "logs": []
}
```

Amazon EC2 Auto Scaling (ASG)

次の例は、Amazon EC2 Auto Scaling (ASG) 用の JSON 形式のコンポーネント設定を示しています。

```
{
    "alarmMetrics" : [
      {
        "alarmMetricName" : "CPUCreditBalance"
      }, {
        "alarmMetricName" : "EBSIOBalance%"
      }
    ],
    "subComponents" : [
      {
        "subComponentType" : "AWS::EC2::Instance",
        "alarmMetrics" : [
          {
            "alarmMetricName" : "CPUUtilization"
          }, {
            "alarmMetricName" : "StatusCheckFailed"
          }
        ],
        "logs" : [
          {
            "logGroupName" : "my_log_group",
            "logPath" : "C:\\LogFolder\\*",
            "logType" : "APPLICATION"
          }
```

```
        ],
        "processes" : [
          {
            "processName" : "my_process",
            "alarmMetrics" : [
              {
                  "alarmMetricName" : "procstat cpu_usage",
                  "monitor" : true
              }, {
                  "alarmMetricName" : "procstat memory_rss",
                  "monitor" : true
              }
            ]
      }
  ],
        "windowsEvents" : [
          {
            "logGroupName" : "my_log_group_2",
            "eventName" : "Application",
            "eventLevels" : [ "ERROR", "WARNING", "CRITICAL" ]
          }
        ]
      }, {
        "subComponentType" : "AWS::EC2::Volume",
        "alarmMetrics" : [
          {
            "alarmMetricName" : "VolumeQueueLength"
          }, {
            "alarmMetricName" : "BurstBalance"
          }
        ]
      }
    ],
    "alarms" : [
      {
        "alarmName" : "my_asg_alarm",
        "severity" : "LOW"
      }
    ]
  }
```

Amazon EKS クラスター

次の例は、Amazon EKS クラスター用の JSON 形式のコンポーネント設定を示しています。

```
{
    "alarmMetrics":[
       {
          "alarmMetricName": "cluster_failed_node_count",
          "monitor":true
       },
       {
          "alarmMetricName": "node_cpu_reserved_capacity",
          "monitor":true
       },
       {
          "alarmMetricName": "node_cpu_utilization",
          "monitor":true
       },
       {
          "alarmMetricName": "node_filesystem_utilization",
          "monitor":true
       },
       {
          "alarmMetricName": "node_memory_reserved_capacity",
          "monitor":true
       },
       {
          "alarmMetricName": "node_memory_utilization",
          "monitor":true
       },
       {
          "alarmMetricName": "node_network_total_bytes",
          "monitor":true
       },
       {
          "alarmMetricName": "pod_cpu_reserved_capacity",
          "monitor":true
       },
       {
          "alarmMetricName": "pod_cpu_utilization",
          "monitor":true
       },
       {
          "alarmMetricName": "pod_cpu_utilization_over_pod_limit",
          "monitor":true
       },
       {
```

```
            "alarmMetricName": "pod_memory_reserved_capacity",
            "monitor":true
         },
         {
            "alarmMetricName": "pod_memory_utilization",
            "monitor":true
         },
         {
            "alarmMetricName": "pod_memory_utilization_over_pod_limit",
            "monitor":true
         },
         {
            "alarmMetricName": "pod_network_rx_bytes",
            "monitor":true
         },
         {
            "alarmMetricName": "pod_network_tx_bytes",
            "monitor":true
         }
      ],
      "logs":[
         {
            "logGroupName": "/aws/containerinsights/kubernetes/application",
            "logType":"APPLICATION",
            "monitor":true,
            "encoding":"utf-8"
         }
      ],
      "subComponents":[
         {
            "subComponentType":"AWS::EC2::Instance",
            "alarmMetrics":[
               {
                  "alarmMetricName":"CPUUtilization",
                  "monitor":true
               },
               {
                  "alarmMetricName":"StatusCheckFailed",
                  "monitor":true
               },
               {
                  "alarmMetricName":"disk_used_percent",
                  "monitor":true
               },
```

```
            {
               "alarmMetricName":"mem_used_percent",
               "monitor":true
            }
         ],
         "logs":[
            {
               "logGroupName":"APPLICATION-KubernetesClusterOnEC2-IAD",
               "logPath":"",
               "logType":"APPLICATION",
               "monitor":true,
               "encoding":"utf-8"
            }
         ],
         "processes" : [
           {
               "processName" : "my_process",
               "alarmMetrics" : [
                   {
                       "alarmMetricName" : "procstat cpu_usage",
                       "monitor" : true
                   }, {
                       "alarmMetricName" : "procstat memory_rss",
                       "monitor" : true
                   }
               ]
           }
       ],
         "windowsEvents":[
            {
               "logGroupName":"my_log_group_2",
               "eventName":"Application",
               "eventLevels":[
                  "ERROR",
                  "WARNING",
                  "CRITICAL"
               ],
               "monitor":true
            }
         ]
      },
      {
         "subComponentType":"AWS::AutoScaling::AutoScalingGroup",
         "alarmMetrics":[
```

```
            {
               "alarmMetricName":"CPUCreditBalance",
               "monitor":true
            },
            {
               "alarmMetricName":"EBSIOBalance%",
               "monitor":true
            }
         ]
      },
      {
         "subComponentType":"AWS::EC2::Volume",
         "alarmMetrics":[
            {
               "alarmMetricName":"VolumeReadBytes",
               "monitor":true
            },
            {
               "alarmMetricName":"VolumeWriteBytes",
               "monitor":true
            },
            {
               "alarmMetricName":"VolumeReadOps",
               "monitor":true
            },
            {
               "alarmMetricName":"VolumeWriteOps",
               "monitor":true
            },
            {
               "alarmMetricName":"VolumeQueueLength",
               "monitor":true
            },
            {
               "alarmMetricName":"BurstBalance",
               "monitor":true
            }
         ]
      }
   ]
}
```

> **Note**
>
> - `AWS::EC2::Instance`、`AWS::EC2::Volume`、`AWS::AutoScaling::AutoScalingGroup` の `subComponents` セクションは、EC2 起動タイプで実行されている Amazon EKS クラスターにのみ適用されます。
> - `subComponents` の `AWS::EC2::Instance` の `windowsEvents` セクションは、Amazon EC2 インスタンスで実行されている Windows にのみ適用されます。

Amazon Elastic Compute Cloud (EC2) インスタンス

次の例は、Amazon EC2 インスタンス用の JSON 形式のコンポーネント設定を示しています。

> **Important**
>
> Amazon EC2 インスタンスが `stopped` 状態になった場合、モニターリングから削除されます。`running` 状態に戻ったら、CloudWatch Application Insights コンソールの [Application details] (アプリケーションの詳細) ページの [Unmonitored components] (モニターリング対象ではないコンポーネント) リストに追加されます。アプリケーションで新しいリソースの自動モニターリングが有効になっている場合、インスタンスは [Monitored components] (モニターリング対象コンポーネント) に追加されます。ただし、ログとメトリクスは、ワークロードのデフォルトに設定されます。以前のログとメトリクスの設定は保存されません。

```
{
    "alarmMetrics" : [
      {
        "alarmMetricName" : "CPUUtilization",
        "monitor" : true
      }, {
        "alarmMetricName" : "StatusCheckFailed"
      }
    ],
    "logs" : [
      {
        "logGroupName" : "my_log_group",
        "logPath" : "C:\\LogFolder\\*",
        "logType" : "APPLICATION",
        "monitor" : true
```

```
      },
      {
        "logGroupName" : "my_log_group_2",
        "logPath" : "C:\\LogFolder2\\*",
        "logType" : "IIS",
        "encoding" : "utf-8"
      }
    ],
    "processes" : [
        {
            "processName" : "my_process",
            "alarmMetrics" : [
                {
                    "alarmMetricName" : "procstat cpu_usage",
                    "monitor" : true
                }, {
                    "alarmMetricName" : "procstat memory_rss",
                    "monitor" : true
                }
            ]
        }
    ],
    "windowsEvents" : [
      {
        "logGroupName" : "my_log_group_3",
        "eventName" : "Application",
        "eventLevels" : [ "ERROR", "WARNING", "CRITICAL" ],
        "monitor" : true
      }, {
        "logGroupName" : "my_log_group_4",
        "eventName" : "System",
        "eventLevels" : [ "ERROR", "WARNING", "CRITICAL" ],
        "monitor" : true
    }],
     "alarms" : [
      {
        "alarmName" : "my_instance_alarm_1",
        "severity" : "HIGH"
      },
      {
        "alarmName" : "my_instance_alarm_2",
        "severity" : "LOW"
      }
    ],
```

```
     "subComponents" : [
     {
       "subComponentType" : "AWS::EC2::Volume",
       "alarmMetrics" : [
       {
         "alarmMetricName" : "VolumeQueueLength",
         "monitor" : "true"
       },
       {
         "alarmMetricName" : "VolumeThroughputPercentage",
         "monitor" : "true"
       },
       {
         "alarmMetricName" : "BurstBalance",
         "monitor" : "true"
       }
    }]
  }
```

Amazon Elastic Container Service (Amazon ECS)

次の例は、Amazon Elastic Container Service (Amazon ECS) 用の JSON 形式のコンポーネント設定を示しています。

```
{
    "alarmMetrics":[
       {
          "alarmMetricName":"CpuUtilized",
          "monitor":true
       },
       {
          "alarmMetricName":"MemoryUtilized",
          "monitor":true
       },
       {
          "alarmMetricName":"NetworkRxBytes",
          "monitor":true
       },
       {
          "alarmMetricName":"NetworkTxBytes",
          "monitor":true
       },
       {
```

```
            "alarmMetricName":"RunningTaskCount",
            "monitor":true
         },
         {
            "alarmMetricName":"PendingTaskCount",
            "monitor":true
         },
         {
            "alarmMetricName":"StorageReadBytes",
            "monitor":true
         },
         {
            "alarmMetricName":"StorageWriteBytes",
            "monitor":true
         }
      ],
      "logs":[
         {
            "logGroupName":"/ecs/my-task-definition",
            "logType":"APPLICATION",
            "monitor":true
         }
      ],
      "subComponents":[
         {
            "subComponentType":"AWS::ElasticLoadBalancing::LoadBalancer",
            "alarmMetrics":[
               {
                  "alarmMetricName":"HTTPCode_Backend_4XX",
                  "monitor":true
               },
               {
                  "alarmMetricName":"HTTPCode_Backend_5XX",
                  "monitor":true
               },
               {
                  "alarmMetricName":"Latency",
                  "monitor":true
               },
               {
                  "alarmMetricName":"SurgeQueueLength",
                  "monitor":true
               },
               {
```

```
                "alarmMetricName":"UnHealthyHostCount",
                "monitor":true
            }
        ]
    },
    {
        "subComponentType":"AWS::ElasticLoadBalancingV2::LoadBalancer",
        "alarmMetrics":[
            {
                "alarmMetricName":"HTTPCode_Target_4XX_Count",
                "monitor":true
            },
            {
                "alarmMetricName":"HTTPCode_Target_5XX_Count",
                "monitor":true
            },
            {
                "alarmMetricName":"TargetResponseTime",
                "monitor":true
            },
            {
                "alarmMetricName":"UnHealthyHostCount",
                "monitor":true
            }
        ]
    },
    {
        "subComponentType":"AWS::EC2::Instance",
        "alarmMetrics":[
            {
                "alarmMetricName":"CPUUtilization",
                "monitor":true
            },
            {
                "alarmMetricName":"StatusCheckFailed",
                "monitor":true
            },
            {
                "alarmMetricName":"disk_used_percent",
                "monitor":true
            },
            {
                "alarmMetricName":"mem_used_percent",
                "monitor":true
```

```
            }
         ],
         "logs":[
            {
               "logGroupName":"my_log_group",
               "logPath":"/mylog/path",
               "logType":"APPLICATION",
               "monitor":true
            }
         ],
         "processes" : [
            {
               "processName" : "my_process",
               "alarmMetrics" : [
                   {
                       "alarmMetricName" : "procstat cpu_usage",
                       "monitor" : true
                   }, {
                       "alarmMetricName" : "procstat memory_rss",
                       "monitor" : true
                   }
                ]
            }
         ],
         "windowsEvents":[
            {
               "logGroupName":"my_log_group_2",
               "eventName":"Application",
               "eventLevels":[
                  "ERROR",
                  "WARNING",
                  "CRITICAL"
               ],
               "monitor":true
            }
         ]
      },
      {
         "subComponentType":"AWS::EC2::Volume",
         "alarmMetrics":[
            {
               "alarmMetricName":"VolumeQueueLength",
               "monitor":"true"
            },
```

```
            {
               "alarmMetricName":"VolumeThroughputPercentage",
               "monitor":"true"
            },
            {
               "alarmMetricName":"BurstBalance",
               "monitor":"true"
            }
         ]
      }
   ]
}
```

> **Note**
>
> - AWS::EC2::Instance および AWS::EC2::Volumeの subComponents セクションは、EC2 起動タイプで実行されている ECS サービスまたは ECS タスクを持つ Amazon ECS クラスターにのみ適用されます。
> - subComponents の AWS::EC2::Instance の windowsEvents セクションは、Amazon EC2 インスタンスで実行されている Windows にのみ適用されます。

Amazon ECS サービス

次の例は、Amazon ECS サービス用の JSON 形式のコンポーネント設定を示しています。

```
{
    "alarmMetrics":[
       {
          "alarmMetricName":"CPUUtilization",
          "monitor":true
       },
       {
          "alarmMetricName":"MemoryUtilization",
          "monitor":true
       },
       {
          "alarmMetricName":"CpuUtilized",
          "monitor":true
       },
       {
```

```
            "alarmMetricName":"MemoryUtilized",
            "monitor":true
         },
         {
            "alarmMetricName":"NetworkRxBytes",
            "monitor":true
         },
         {
            "alarmMetricName":"NetworkTxBytes",
            "monitor":true
         },
         {
            "alarmMetricName":"RunningTaskCount",
            "monitor":true
         },
         {
            "alarmMetricName":"PendingTaskCount",
            "monitor":true
         },
         {
            "alarmMetricName":"StorageReadBytes",
            "monitor":true
         },
         {
            "alarmMetricName":"StorageWriteBytes",
            "monitor":true
         }
      ],
      "logs":[
         {
            "logGroupName":"/ecs/my-task-definition",
            "logType":"APPLICATION",
            "monitor":true
         }
      ],
      "subComponents":[
         {
            "subComponentType":"AWS::ElasticLoadBalancing::LoadBalancer",
            "alarmMetrics":[
               {
                  "alarmMetricName":"HTTPCode_Backend_4XX",
                  "monitor":true
               },
               {
```

```
                "alarmMetricName":"HTTPCode_Backend_5XX",
                "monitor":true
             },
             {
                "alarmMetricName":"Latency",
                "monitor":true
             },
             {
                "alarmMetricName":"SurgeQueueLength",
                "monitor":true
             },
             {
                "alarmMetricName":"UnHealthyHostCount",
                "monitor":true
             }
          ]
       },
       {
          "subComponentType":"AWS::ElasticLoadBalancingV2::LoadBalancer",
          "alarmMetrics":[
             {
                "alarmMetricName":"HTTPCode_Target_4XX_Count",
                "monitor":true
             },
             {
                "alarmMetricName":"HTTPCode_Target_5XX_Count",
                "monitor":true
             },
             {
                "alarmMetricName":"TargetResponseTime",
                "monitor":true
             },
             {
                "alarmMetricName":"UnHealthyHostCount",
                "monitor":true
             }
          ]
       },
       {
          "subComponentType":"AWS::EC2::Instance",
          "alarmMetrics":[
             {
                "alarmMetricName":"CPUUtilization",
                "monitor":true
```

```
            },
            {
               "alarmMetricName":"StatusCheckFailed",
               "monitor":true
            },
            {
               "alarmMetricName":"disk_used_percent",
               "monitor":true
            },
            {
               "alarmMetricName":"mem_used_percent",
               "monitor":true
            }
         ],
         "logs":[
            {
               "logGroupName":"my_log_group",
               "logPath":"/mylog/path",
               "logType":"APPLICATION",
               "monitor":true
            }
         ],
         "processes" : [
            {
               "processName" : "my_process",
               "alarmMetrics" : [
               {
                   "alarmMetricName" : "procstat cpu_usage",
                   "monitor" : true
               }, {
                   "alarmMetricName" : "procstat memory_rss",
                   "monitor" : true
               }
           ]
       }
   ],
         "windowsEvents":[
            {
               "logGroupName":"my_log_group_2",
               "eventName":"Application",
               "eventLevels":[
                  "ERROR",
                  "WARNING",
                  "CRITICAL"
```

```
                ],
                "monitor":true
            }
         ]
      },
      {
         "subComponentType":"AWS::EC2::Volume",
         "alarmMetrics":[
            {
               "alarmMetricName":"VolumeQueueLength",
               "monitor":"true"
            },
            {
               "alarmMetricName":"VolumeThroughputPercentage",
               "monitor":"true"
            },
            {
               "alarmMetricName":"BurstBalance",
               "monitor":"true"
            }
         ]
      }
   ]
}
```

> **Note**
>
> - `AWS::EC2::Instance` および `AWS::EC2::Volume` の `subComponents` セクションは、EC2 起動タイプで実行されている Amazon ECS にのみ適用されます。
> - `subComponents` の `AWS::EC2::Instance` の `windowsEvents` セクションは、Amazon EC2 インスタンスで実行されている Windows にのみ適用されます。

Amazon ECS タスク

次の例は、Amazon ECS タスク用の JSON 形式のコンポーネント設定を示しています。

```
{
    "logs":[
       {
          "logGroupName":"/ecs/my-task-definition",
```

```
          "logType":"APPLICATION",
          "monitor":true
       }
    ],
    "processes" : [
        {
            "processName" : "my_process",
            "alarmMetrics" : [
                {
                    "alarmMetricName" : "procstat cpu_usage",
                    "monitor" : true
                }, {
                    "alarmMetricName" : "procstat memory_rss",
                    "monitor" : true
                }
            ]
        }
    ]
 }
```

Amazon Elastic File System (Amazon EFS)

次の例は、Amazon EFS 用の JSON 形式のコンポーネント設定を示しています。

```
{
   "alarmMetrics": [
     {
       "alarmMetricName": "BurstCreditBalance",
       "monitor": true
     },
     {
       "alarmMetricName": "PercentIOLimit",
       "monitor": true
     },
     {
       "alarmMetricName": "PermittedThroughput",
       "monitor": true
     },
     {
       "alarmMetricName": "MeteredIOBytes",
       "monitor": true
     },
     {
       "alarmMetricName": "TotalIOBytes",
```

```
        "monitor": true
      },
      {
        "alarmMetricName": "DataWriteIOBytes",
        "monitor": true
      },
      {
        "alarmMetricName": "DataReadIOBytes",
        "monitor": true
      },
      {
        "alarmMetricName": "MetadataIOBytes",
        "monitor": true
      },
      {
        "alarmMetricName": "ClientConnections",
        "monitor": true
      },
      {
        "alarmMetricName": "TimeSinceLastSync",
        "monitor": true
      },
      {
        "alarmMetricName": "Throughput",
        "monitor": true
      },
      {
        "alarmMetricName": "PercentageOfPermittedThroughputUtilization",
        "monitor": true
      },
      {
        "alarmMetricName": "ThroughputIOPS",
        "monitor": true
      },
      {
        "alarmMetricName": "PercentThroughputDataReadIOBytes",
        "monitor": true
      },
      {
        "alarmMetricName": "PercentThroughputDataWriteIOBytes",
        "monitor": true
      },
      {
        "alarmMetricName": "PercentageOfIOPSDataReadIOBytes",
```

```
        "monitor": true
      },
      {
        "alarmMetricName": "PercentageOfIOPSDataWriteIOBytes",
        "monitor": true
      },
      {
        "alarmMetricName": "AverageDataReadIOBytesSize",
        "monitor": true
      },
      {
        "alarmMetricName": "AverageDataWriteIOBytesSize",
        "monitor": true
      }
    ],
    "logs": [
     {
     "logGroupName": "/aws/efs/utils",
     "logType": "EFS_MOUNT_STATUS",
     "monitor": true,
     }
    ]
 }
```

Amazon FSx

次の例は、Amazon FSx 用の JSON 形式のコンポーネント設定を示しています。

```
{
  "alarmMetrics": [
    {
      "alarmMetricName": "DataReadBytes",
      "monitor": true
    },
    {
      "alarmMetricName": "DataWriteBytes",
      "monitor": true
    },
    {
      "alarmMetricName": "DataReadOperations",
      "monitor": true
    },
    {
      "alarmMetricName": "DataWriteOperations",
```

```
      "monitor": true
    },
    {
      "alarmMetricName": "MetadataOperations",
      "monitor": true
    },
    {
      "alarmMetricName": "FreeStorageCapacity",
      "monitor": true
    }
  ]
}
```

Amazon Relational Database Service (RDS) Aurora MySQL

次の例は、Amazon RDS Aurora MySQL 用の JSON 形式のコンポーネント設定を示しています。

```
{
  "alarmMetrics": [
    {
      "alarmMetricName": "CPUUtilization",
      "monitor": true
    },
    {
      "alarmMetricName": "CommitLatency",
      "monitor": true
    }
  ],
  "logs": [
    {
      "logType": "MYSQL",
      "monitor": true,
    },
    {
      "logType": "MYSQL_SLOW_QUERY",
      "monitor": false
    }
  ]
}
```

Amazon Relational Database Service (RDS) インスタンス

次の例は、Amazon RDS インスタンス用の JSON 形式のコンポーネント設定を示しています。

```
{
  "alarmMetrics" : [
    {
      "alarmMetricName" : "BurstBalance",
      "monitor" : true
    }, {
      "alarmMetricName" : "WriteThroughput",
      "monitor" : false
    }
  ],

  "alarms" : [
    {
      "alarmName" : "my_rds_instance_alarm",
      "severity" : "MEDIUM"
    }
  ]
}
```

Amazon Route 53 ヘルスチェック

次の例は、Amazon Route 53 ヘルスチェック用の JSON 形式のコンポーネント設定を示しています。

```
{
  "alarmMetrics": [
    {
      "alarmMetricName": "ChildHealthCheckHealthyCount",
      "monitor": true
    },
    {
      "alarmMetricName": "ConnectionTime",
      "monitor": true
    },
    {
      "alarmMetricName": "HealthCheckPercentageHealthy",
      "monitor": true
    },
    {
      "alarmMetricName": "HealthCheckStatus",
      "monitor": true
    },
    {
```

```
      "alarmMetricName": "SSLHandshakeTime",
      "monitor": true
    },
    {
      "alarmMetricName": "TimeToFirstByte",
      "monitor": true
    }
  ]
}
```

Amazon Route 53 ホストゾーン

次の例は、Amazon Route 53 ホストゾーン用の JSON 形式のコンポーネント設定を示しています。

```
{
  "alarmMetrics": [
    {
      "alarmMetricName": "DNSQueries",
      "monitor": true
    },
    {
      "alarmMetricName": "DNSSECInternalFailure",
      "monitor": true
    },
    {
      "alarmMetricName": "DNSSECKeySigningKeysNeedingAction",
      "monitor": true
    },
    {
      "alarmMetricName": "DNSSECKeySigningKeyMaxNeedingActionAge",
      "monitor": true
    },
    {
      "alarmMetricName": "DNSSECKeySigningKeyAge",
      "monitor": true
    }
  ],
  "logs": [
    {
      "logGroupName":"/hosted-zone/logs",
      "logType": "ROUTE53_DNS_PUBLIC_QUERY_LOGS",
      "monitor": true
    }
  ]
```

```
}
```

Amazon Route 53 Resolver エンドポイント

次の例は、Amazon Route 53 Resolver エンドポイント用の JSON 形式のコンポーネント設定を示しています。

```
{
  "alarmMetrics": [
    {
      "alarmMetricName": "EndpointHealthyENICount",
      "monitor": true
    },
    {
      "alarmMetricName": "EndpointUnHealthyENICount",
      "monitor": true
    },
    {
      "alarmMetricName": "InboundQueryVolume",
      "monitor": true
    },
    {
      "alarmMetricName": "OutboundQueryVolume",
      "monitor": true
    },
    {
      "alarmMetricName": "OutboundQueryAggregateVolume",
      "monitor": true
    }
  ]
}
```

Amazon Route 53 Resolver クエリログ記録の設定

次の例は、Amazon Route 53 Resolver クエリログ記録設定用の JSON 形式のコンポーネント設定を示しています。

```
{
  "logs": [
    {
      "logGroupName": "/resolver-query-log-config/logs",
      "logType": "ROUTE53_RESOLVER_QUERY_LOGS",
      "monitor": true
```

```
    }
  ]
}
```

Amazon S3 バケット

次の例は、Amazon S3 バケット用の JSON 形式のコンポーネント設定を示しています。

```
{
    "alarmMetrics" : [
        {
            "alarmMetricName" : "ReplicationLatency",
            "monitor" : true
        }, {
            "alarmMetricName" : "5xxErrors",
            "monitor" : true
        }, {
            "alarmMetricName" : "BytesDownloaded"
            "monitor" : true
        }
    ]
}
```

Amazon Simple Queue Service (SQS)

次の例は、Amazon Simple Queue Service 用の JSON 形式のコンポーネント設定を示しています。

```
{
  "alarmMetrics" : [
    {
      "alarmMetricName" : "ApproximateAgeOfOldestMessage"
    }, {
      "alarmMetricName" : "NumberOfEmptyReceives"
    }
  ],
  "alarms" : [
    {
      "alarmName" : "my_sqs_alarm",
      "severity" : "MEDIUM"
    }
  ]
}
```

Amazon SNS トピック

次の例は、Amazon SNS トピック用の JSON 形式のコンポーネント設定を示しています。

```
{
  "alarmMetrics": [
    {
      "alarmMetricName": "NumberOfNotificationsFailed",
      "monitor": true
    },
    {
      "alarmMetricName": "NumberOfNotificationsFilteredOut-InvalidAttributes",
      "monitor": true
    },
    {
      "alarmMetricName": "NumberOfNotificationsFilteredOut-NoMessageAttributes",
      "monitor": true
    },
    {
      "alarmMetricName": "NumberOfNotificationsFailedToRedriveToDlq",
      "monitor": true
    }
  ]
}
```

Amazon Virtual Private Cloud (Amazon VPC)

次の例は、Amazon VPC 用の JSON 形式のコンポーネント設定を示しています。

```
{
  "alarmMetrics": [
    {
      "alarmMetricName": "NetworkAddressUsage",
      "monitor": true
    },
    {
      "alarmMetricName": "NetworkAddressUsagePeered",
      "monitor": true
    },
    {
      "alarmMetricName": "VPCFirewallQueryVolume",
      "monitor": true
    }
```

```
  ]
}
```

Amazon VPC ネットワークアドレス変換 (NAT) ゲートウェイ

次の例は、NAT ゲートウェイ用の JSON 形式のコンポーネント設定を示しています。

```
{
  "alarmMetrics": [
    {
      "alarmMetricName": "ErrorPortAllocation",
      "monitor": true
    },
    {
      "alarmMetricName": "IdleTimeoutCount",
      "monitor": true
    }
  ]
}
```

API Gateway REST API ステージ

次の例は、API Gateway REST API ステージ用の JSON 形式のコンポーネント設定を示しています。

```
{
     "alarmMetrics" : [
         {
             "alarmMetricName" : "4XXError",
             "monitor" : true
         },
         {
             "alarmMetricName" : "5XXError",
             "monitor" : true
         }
     ],
    "logs" : [
        {
            "logType" : "API_GATEWAY_EXECUTION",
            "monitor" : true
        },
        {
            "logType" : "API_GATEWAY_ACCESS",
```

```
            "monitor" : true
        }
    ]
}
```

Application Elastic Load Balancing

次の例は、Application Elastic Load Balancing 用の JSON 形式のコンポーネント設定を示しています。

```
{
  "alarmMetrics": [
    {
      "alarmMetricName": "ActiveConnectionCount",
    }, {
      "alarmMetricName": "TargetResponseTime"
    }
  ],
  "subComponents": [
    {
      "subComponentType": "AWS::EC2::Instance",
      "alarmMetrics": [
        {
          "alarmMetricName": "CPUUtilization",
        }, {
          "alarmMetricName": "StatusCheckFailed"
        }
      ],
      "logs": [
        {
          "logGroupName": "my_log_group",
          "logPath": "C:\\LogFolder\\*",
          "logType": "APPLICATION",
        }
      ],
      "windowsEvents": [
        {
          "logGroupName": "my_log_group_2",
          "eventName": "Application",
          "eventLevels": [ "ERROR", "WARNING", "CRITICAL" ]
        }
      ]
    }, {
```

```
      "subComponentType": "AWS::EC2::Volume",
      "alarmMetrics": [
        {
          "alarmMetricName": "VolumeQueueLength",
        }, {
          "alarmMetricName": "BurstBalance"
        }
      ]
    }
  ],

  "alarms": [
    {
      "alarmName": "my_alb_alarm",
      "severity": "LOW"
    }
  ]
}
```

AWS Lambda 関数

次の例は、AWS Lambda 機能用の JSON 形式のコンポーネント設定を示しています。

```
{
  "alarmMetrics": [
    {
      "alarmMetricName": "Errors",
      "monitor": true
    },
    {
      "alarmMetricName": "Throttles",
      "monitor": true
    },
    {
      "alarmMetricName": "IteratorAge",
      "monitor": true
    },
    {
      "alarmMetricName": "Duration",
      "monitor": true
    }
  ],
  "logs": [
    {
```

```
      "logType": "DEFAULT",
      "monitor": true
    }
  ]
}
```

AWS Network Firewall ルールグループ

次の例は、AWS Network Firewall ルールグループ用の JSON 形式のコンポーネント設定を示しています。

```
{
  "alarmMetrics": [
    {
      "alarmMetricName": "FirewallRuleGroupQueryVolume",
      "monitor": true
    }
  ]
}
```

AWS Network Firewall ルールグループ関連付け

次の例は、AWS Network Firewall ルールグループ関連付け用の JSON 形式のコンポーネント設定を示しています。

```
{
  "alarmMetrics": [
    {
      "alarmMetricName": "FirewallRuleGroupQueryVolume",
      "monitor": true
    }
  ]
}
```

AWS Step Functions

次の例は、AWS Step Functions 用の JSON 形式のコンポーネント設定を示しています。

```
{
  "alarmMetrics": [
    {
      "alarmMetricName": "ExecutionsFailed",
      "monitor": true
```

```
    },
    {
      "alarmMetricName": "LambdaFunctionsFailed",
      "monitor": true
    },
    {
      "alarmMetricName": "ProvisionedRefillRate",
      "monitor": true
    }
  ],
  "logs": [
    {
     "logGroupName": "/aws/states/HelloWorld-Logs",
      "logType": "STEP_FUNCTION",
      "monitor": true,
    }
  ]
}
```

カスタマーグループ化された Amazon EC2 インスタンス

次の例は、カスタマーグループ化された Amazon EC2 インスタンス用の JSON 形式のコンポーネント設定を示しています。

```
{
    "subComponents": [
      {
        "subComponentType": "AWS::EC2::Instance",
        "alarmMetrics": [
          {
            "alarmMetricName": "CPUUtilization",
          },
          {
            "alarmMetricName": "StatusCheckFailed"
          }
        ],
        "logs": [
          {
            "logGroupName": "my_log_group",
            "logPath": "C:\\LogFolder\\*",
            "logType": "APPLICATION",
          }
        ],
```

```
        "processes": [
            {
                "processName": "my_process",
                "alarmMetrics": [
                    {
                        "alarmMetricName": "procstat cpu_usage",
                        "monitor": true
                    }, {
                        "alarmMetricName": "procstat memory_rss",
                        "monitor": true
                    }
                ]
            }
        ],
        "windowsEvents": [
          {
            "logGroupName": "my_log_group_2",
            "eventName": "Application",
            "eventLevels": [ "ERROR", "WARNING", "CRITICAL" ]
          }
        ]
      }, {
        "subComponentType": "AWS::EC2::Volume",
        "alarmMetrics": [
          {
            "alarmMetricName": "VolumeQueueLength",
          }, {
            "alarmMetricName": "BurstBalance"
          }
        ]
      }
    ],
    "alarms": [
      {
        "alarmName": "my_alarm",
        "severity": "MEDIUM"
      }
    ]
  }
```

Elastic Load Balancing

次の例は、Elastic Load Balancing 用の JSON 形式のコンポーネント設定を示しています。

```
{
    "alarmMetrics": [
      {
        "alarmMetricName": "EstimatedALBActiveConnectionCount"
      }, {
        "alarmMetricName": "HTTPCode_Backend_5XX"
      }
    ],
    "subComponents": [
      {
        "subComponentType": "AWS::EC2::Instance",
        "alarmMetrics": [
          {
            "alarmMetricName": "CPUUtilization"
          }, {
            "alarmMetricName": "StatusCheckFailed"
          }
        ],
        "logs": [
          {
            "logGroupName": "my_log_group",
            "logPath": "C:\\LogFolder\\*",
            "logType": "APPLICATION"
          }
        ],
        "processes": [
          {
            "processName": "my_process",
            "alarmMetrics": [
                {
                  "alarmMetricName": "procstat cpu_usage",
                  "monitor": true
                }, {
                  "alarmMetricName": "procstat memory_rss",
                  "monitor": true
                }
             ]
          }
        ],
        "windowsEvents": [
          {
            "logGroupName": "my_log_group_2",
            "eventName": "Application",
```

```
            "eventLevels": [ "ERROR", "WARNING", "CRITICAL" ],
            "monitor": true
          }
        ]
      }, {
        "subComponentType": "AWS::EC2::Volume",
        "alarmMetrics": [
          {
            "alarmMetricName": "VolumeQueueLength"
          }, {
            "alarmMetricName": "BurstBalance"
          }
        ]
      }
    ],

    "alarms": [
      {
        "alarmName": "my_elb_alarm",
        "severity": "HIGH"
      }
    ]
  }
```

Java

次の例は、Java 用の JSON 形式のコンポーネント設定を示しています。

```
{
  "alarmMetrics": [ {
    "alarmMetricName": "java_lang_threading_threadcount",
    "monitor": true
  },
  {
    "alarmMetricName": "java_lang_memory_heapmemoryusage_used",
    "monitor": true
  },
  {
    "alarmMetricName": "java_lang_memory_heapmemoryusage_committed",
    "monitor": true
  }],
  "logs": [ ],
  "JMXPrometheusExporter": {
      "hostPort": "8686",
```

```
      "prometheusPort": "9404"
  }
}
```

> **Note**
>
> Application Insights は、Prometheus JMX Exporter の認証の設定をサポートしていません。認証の設定方法については、Prometheus JMX Exporter の設定例をご参照ください。

Amazon EC2 での Kubernetes

次の例は、Amazon EC2 での Kubernetes 用の JSON 形式のコンポーネント設定を示しています。

```
{
    "alarmMetrics":[
       {
          "alarmMetricName":"cluster_failed_node_count",
          "monitor":true
       },
       {
          "alarmMetricName":"node_cpu_reserved_capacity",
          "monitor":true
       },
       {
          "alarmMetricName":"node_cpu_utilization",
          "monitor":true
       },
       {
          "alarmMetricName":"node_filesystem_utilization",
          "monitor":true
       },
       {
          "alarmMetricName":"node_memory_reserved_capacity",
          "monitor":true
       },
       {
          "alarmMetricName":"node_memory_utilization",
          "monitor":true
       },
       {
          "alarmMetricName":"node_network_total_bytes",
          "monitor":true
```

```
        },
        {
           "alarmMetricName":"pod_cpu_reserved_capacity",
           "monitor":true
        },
        {
           "alarmMetricName":"pod_cpu_utilization",
           "monitor":true
        },
        {
           "alarmMetricName":"pod_cpu_utilization_over_pod_limit",
           "monitor":true
        },
        {
           "alarmMetricName":"pod_memory_reserved_capacity",
           "monitor":true
        },
        {
           "alarmMetricName":"pod_memory_utilization",
           "monitor":true
        },
        {
           "alarmMetricName":"pod_memory_utilization_over_pod_limit",
           "monitor":true
        },
        {
           "alarmMetricName":"pod_network_rx_bytes",
           "monitor":true
        },
        {
           "alarmMetricName":"pod_network_tx_bytes",
           "monitor":true
        }
     ],
     "logs":[
        {
           "logGroupName":"/aws/containerinsights/kubernetes/application",
           "logType":"APPLICATION",
           "monitor":true,
           "encoding":"utf-8"
        }
     ],
     "subComponents":[
        {
```

```
          "subComponentType":"AWS::EC2::Instance",
          "alarmMetrics":[
             {
                "alarmMetricName":"CPUUtilization",
                "monitor":true
             },
             {
                "alarmMetricName":"StatusCheckFailed",
                "monitor":true
             },
             {
                "alarmMetricName":"disk_used_percent",
                "monitor":true
             },
             {
                "alarmMetricName":"mem_used_percent",
                "monitor":true
             }
          ],
          "logs":[
             {
                "logGroupName":"APPLICATION-KubernetesClusterOnEC2-IAD",
                "logPath":"",
                "logType":"APPLICATION",
                "monitor":true,
                "encoding":"utf-8"
             }
          ],
          "processes" : [
            {
                "processName" : "my_process",
                "alarmMetrics" : [
                    {
                        "alarmMetricName" : "procstat cpu_usage",
                        "monitor" : true
                    }, {
                        "alarmMetricName" : "procstat memory_rss",
                        "monitor" : true
                    }
                ]
            }
        ]
       },
       {
```

```
          "subComponentType":"AWS::EC2::Volume",
          "alarmMetrics":[
             {
                "alarmMetricName":"VolumeReadBytes",
                "monitor":true
             },
             {
                "alarmMetricName":"VolumeWriteBytes",
                "monitor":true
             },
             {
                "alarmMetricName":"VolumeReadOps",
                "monitor":true
             },
             {
                "alarmMetricName":"VolumeWriteOps",
                "monitor":true
             },
             {
                "alarmMetricName":"VolumeQueueLength",
                "monitor":true
             },
             {
                "alarmMetricName":"BurstBalance",
                "monitor":true
             }
          ]
       }
    ]
 }
```

RDS MariaDB と RDS MySQL

次の例は、RDS MariaDB と RDS MySQL 用の JSON 形式のコンポーネント設定を示しています。

```
{
  "alarmMetrics": [
    {
      "alarmMetricName": "CPUUtilization",
      "monitor": true
    }
  ],
  "logs": [
    {
```

```
      "logType": "MYSQL",
      "monitor": true,
    },
    {
      "logType": "MYSQL_SLOW_QUERY",
      "monitor": false
    }
  ]
}
```

RDS Oracle

次の例は、RDS Oracle 用の JSON 形式のコンポーネント設定を示しています。

```
{
  "alarmMetrics": [
    {
      "alarmMetricName": "CPUUtilization",
      "monitor": true
    }
  ],
  "logs": [
    {
      "logType": "ORACLE_ALERT",
      "monitor": true,
    },
    {
      "logType": "ORACLE_LISTENER",
      "monitor": false
    }
  ]
}
```

RDS PostgreSQL

次の例は、RDS PostgreSQL 用の JSON 形式のコンポーネント設定を示しています。

```
{
  "alarmMetrics": [
    {
      "alarmMetricName": "CPUUtilization",
      "monitor": true
    }
```

```
  ],
  "logs": [
    {
      "logType": "POSTGRESQL",
      "monitor": true
    }
  ]
}
```

Amazon EC2 での SAP ASE

次の例は、Amazon EC2 での SAP ASE 用の JSON 形式のコンポーネント設定を示しています。

```
{
  "subComponents": [
    {
      "subComponentType": "AWS::EC2::Instance",
      "alarmMetrics": [
        {
          "alarmMetricName": "asedb_database_availability",
          "monitor": true
        },
        {
          "alarmMetricName": "asedb_trunc_log_on_chkpt_enabled",
          "monitor": true
        },
        {
          "alarmMetricName": "asedb_last_db_backup_age_in_days",
          "monitor": true
        },
        {
          "alarmMetricName": "asedb_last_transaction_log_backup_age_in_hours",
          "monitor": true
        },
        {
          "alarmMetricName": "asedb_suspected_database",
          "monitor": true
        },
        {
          "alarmMetricName": "asedb_db_space_usage_percent",
          "monitor": true
        },
        {
          "alarmMetricName": "asedb_db_log_space_usage_percent",
```

```
          "monitor": true
        },
        {
          "alarmMetricName": "asedb_locked_login",
          "monitor": true
        },
        {
          "alarmMetricName": "asedb_data_cache_hit_ratio",
          "monitor": true
        }
      ],
      "logs": [
        {
          "logGroupName": "SAP_ASE_SERVER_LOGS-my-resource-group",
          "logPath": "/sybase/SY2/ASE-*/install/SY2.log",
          "logType": "SAP_ASE_SERVER_LOGS",
          "monitor": true,
          "encoding": "utf-8"
        },
        {
          "logGroupName": "SAP_ASE_BACKUP_SERVER_LOGS-my-resource-group",
          "logPath": "/sybase/SY2/ASE-*/install/SY2_BS.log",
          "logType": "SAP_ASE_BACKUP_SERVER_LOGS",
          "monitor": true,
          "encoding": "utf-8"
        }
  ],
  "sapAsePrometheusExporter": {
    "sapAseSid": "ASE",
    "sapAsePort": "4901",
    "sapAseSecretName": "ASE_DB_CREDS",
    "prometheusPort": "9399",
    "agreeToEnableASEMonitoring": true
}
```

Amazon EC2 での SAP ASE High Availability

次の例は、Amazon EC2 での SAP ASE High Availability 用の JSON 形式のコンポーネント設定を示しています。

```
{
  "subComponents": [
    {
```

```
      "subComponentType": "AWS::EC2::Instance",
      "alarmMetrics": [
        {
          "alarmMetricName": "asedb_database_availability",
          "monitor": true
        },
        {
          "alarmMetricName": "asedb_trunc_log_on_chkpt_enabled",
          "monitor": true
        },
        {
          "alarmMetricName": "asedb_last_db_backup_age_in_days",
          "monitor": true
        },
        {
          "alarmMetricName": "asedb_last_transaction_log_backup_age_in_hours",
          "monitor": true
        },
        {
          "alarmMetricName": "asedb_suspected_database",
          "monitor": true
        },
        {
          "alarmMetricName": "asedb_db_space_usage_percent",
          "monitor": true
        },
        {
          "alarmMetricName": "asedb_ha_replication_state",
          "monitor": true
        },
        {
          "alarmMetricName": "asedb_ha_replication_mode",
          "monitor": true
        },
        {
          "alarmMetricName": "asedb_ha_replication_latency_in_minutes",
          "monitor": true
        }
      ],
      "logs": [
        {
          "logGroupName": "SAP_ASE_SERVER_LOGS-my-resource-group",
          "logPath": "/sybase/SY2/ASE-*/install/SY2.log",
          "logType": "SAP_ASE_SERVER_LOGS",
```

```
          "monitor": true,
          "encoding": "utf-8"
        },
        {
          "logGroupName": "SAP_ASE_BACKUP_SERVER_LOGS-my-resource-group",
          "logPath": "/sybase/SY2/ASE-*/install/SY2_BS.log",
          "logType": "SAP_ASE_BACKUP_SERVER_LOGS",
          "monitor": true,
          "encoding": "utf-8"
        },
        {
          "logGroupName": "SAP_ASE_REP_SERVER_LOGS-my-resource-group",
          "logPath": "/sybase/SY2/DM/repservername/repservername.log",
          "logType": "SAP_ASE_REP_SERVER_LOGS",
          "monitor": true,
          "encoding": "utf-8"
        },
        {
          "logGroupName": "SAP_ASE_RMA_AGENT_LOGS-my-resource-group",
          "logPath": "/sybase/SY2/DM/RMA-*/instances/AgentContainer/logs/",
          "logType": "SAP_ASE_RMA_AGENT_LOGS",
          "monitor": true,
          "encoding": "utf-8"
        },
        {
          "logGroupName": "SAP_ASE_FAULT_MANAGER_LOGS-my-resource-group",
          "logPath": "/opt/sap/FaultManager/dev_sybdbfm",
          "logType": "SAP_ASE_FAULT_MANAGER_LOGS",
          "monitor": true,
          "encoding": "utf-8"
        }
  ],
  "sapAsePrometheusExporter": {
    "sapAseSid": "ASE",
    "sapAsePort": "4901",
    "sapAseSecretName": "ASE_DB_CREDS",
    "prometheusPort": "9399",
    "agreeToEnableASEMonitoring": true
}
```

Amazon EC2 での SAP HANA

次の例は、Amazon EC2 での SAP HANA 用の JSON 形式のコンポーネント設定を示しています。

```
{
  "subComponents": [
    {
      "subComponentType": "AWS::EC2::Instance",
      "alarmMetrics": [
        {
          "alarmMetricName": "hanadb_server_startup_time_variations_seconds",
          "monitor": true
        },
        {
          "alarmMetricName": "hanadb_level_5_alerts_count",
          "monitor": true
        },
        {
          "alarmMetricName": "hanadb_level_4_alerts_count",
          "monitor": true
        },
        {
          "alarmMetricName": "hanadb_out_of_memory_events_count",
          "monitor": true
        },
        {
          "alarmMetricName": "hanadb_max_trigger_read_ratio_percent",
          "monitor": true
        },
        {
          "alarmMetricName": "hanadb_table_allocation_limit_used_percent",
          "monitor": true
        },
        {
          "alarmMetricName": "hanadb_cpu_usage_percent",
          "monitor": true
        },
        {
          "alarmMetricName": "hanadb_plan_cache_hit_ratio_percent",
          "monitor": true
        },
        {
          "alarmMetricName": "hanadb_last_data_backup_age_days",
          "monitor": true
        }
      ],
      "logs": [
```

```
        {
          "logGroupName": "SAP_HANA_TRACE-my-resourge-group",
          "logPath": "/usr/sap/HDB/HDB00/*/trace/*.trc",
          "logType": "SAP_HANA_TRACE",
          "monitor": true,
          "encoding": "utf-8"
        },
        {
          "logGroupName": "SAP_HANA_LOGS-my-resource-group",
          "logPath": "/usr/sap/HDB/HDB00/*/trace/*.log",
          "logType": "SAP_HANA_LOGS",
          "monitor": true,
          "encoding": "utf-8"
        }
      ]
    }
  ],
  "hanaPrometheusExporter": {
    "hanaSid": "HDB",
    "hanaPort": "30013",
    "hanaSecretName": "HANA_DB_CREDS",
    "prometheusPort": "9668"
  }
}
```

Amazon EC2 での SAP HANA High Availability

次の例は、Amazon EC2 での SAP HANA High Availability 用の JSON 形式のコンポーネント設定を示しています。

```
{
  "subComponents": [
    {
      "subComponentType": "AWS::EC2::Instance",
      "alarmMetrics": [
        {
          "alarmMetricName": "hanadb_server_startup_time_variations_seconds",
          "monitor": true
        },
        {
          "alarmMetricName": "hanadb_level_5_alerts_count",
          "monitor": true
        },
```

```
        {
          "alarmMetricName": "hanadb_level_4_alerts_count",
          "monitor": true
        },
        {
          "alarmMetricName": "hanadb_out_of_memory_events_count",
          "monitor": true
        },
        {
          "alarmMetricName": "ha_cluster_pacemaker_stonith_enabled",
          "monitor": true
        }
      ],
      "logs": [
        {
          "logGroupName": "SAP_HANA_TRACE-my-resourge-group",
          "logPath": "/usr/sap/HDB/HDB00/*/trace/*.trc",
          "logType": "SAP_HANA_TRACE",
          "monitor": true,
          "encoding": "utf-8"
        },
        {
          "logGroupName": "SAP_HANA_HIGH_AVAILABILITY-my-resource-group",
          "logPath": "/var/log/pacemaker/pacemaker.log",
          "logType": "SAP_HANA_HIGH_AVAILABILITY",
          "monitor": true,
          "encoding": "utf-8"
        }
      ]
    }
  ],
  "hanaPrometheusExporter": {
    "hanaSid": "HDB",
    "hanaPort": "30013",
    "hanaSecretName": "HANA_DB_CREDS",
    "prometheusPort": "9668"
  },
  "haClusterPrometheusExporter": {
    "prometheusPort": "9664"
  }
}
```

Amazon EC2 での SAP NetWeaver

次の例は、Amazon EC2 での SAP Netweaver 用の JSON 形式のコンポーネント設定を示しています。

```
{
  "subComponents": [
    {
      "subComponentType": "AWS::EC2::Instance",
      "alarmMetrics": [
        {
          "alarmMetricName": "CPUUtilization",
          "monitor": true
        },
        {
          "alarmMetricName": "StatusCheckFailed",
          "monitor": true
        },
        {
          "alarmMetricName": "disk_used_percent",
          "monitor": true
        },
        {
          "alarmMetricName": "mem_used_percent",
          "monitor": true
        },
        {
          "alarmMetricName": "sap_alerts_ResponseTime",
          "monitor": true
        },
        {
          "alarmMetricName": "sap_alerts_ResponseTimeDialog",
          "monitor": true
        },
        {
          "alarmMetricName": "sap_alerts_ResponseTimeDialogRFC",
          "monitor": true
        },
        {
          "alarmMetricName": "sap_alerts_DBRequestTime",
          "monitor": true
        },
        {
          "alarmMetricName": "sap_alerts_LongRunners",
```

```
          "monitor": true
        },
        {
          "alarmMetricName": "sap_alerts_AbortedJobs",
          "monitor": true
        },
        {
          "alarmMetricName": "sap_alerts_BasisSystem",
          "monitor": true
        },
        {
          "alarmMetricName": "sap_alerts_Database",
          "monitor": true
        },
        {
          "alarmMetricName": "sap_alerts_Security",
          "monitor": true
        },
        {
          "alarmMetricName": "sap_alerts_System",
          "monitor": true
        },
        {
          "alarmMetricName": "sap_alerts_QueueTime",
          "monitor": true
        },
        {
          "alarmMetricName": "sap_alerts_Availability",
          "monitor": true
        },
        {
          "alarmMetricName": "sap_start_service_processes",
          "monitor": true
        },
        {
          "alarmMetricName": "sap_dispatcher_queue_now",
          "monitor": true
        },
        {
          "alarmMetricName": "sap_dispatcher_queue_max",
          "monitor": true
        },
        {
          "alarmMetricName": "sap_enqueue_server_locks_max",
```

```
          "monitor": true
        },
        {
          "alarmMetricName": "sap_enqueue_server_locks_now",
          "monitor": true
        },
        {
          "alarmMetricName": "sap_enqueue_server_locks_state",
          "monitor": true
        },
        {
          "alarmMetricName": "sap_enqueue_server_replication_state",
          "monitor": true
        }
      ],
      "logs": [
        {
          "logGroupName": "SAP_NETWEAVER_DEV_TRACE_LOGS-NetWeaver-ML4",
          "logPath": "/usr/sap/ML4/*/work/dev_w*",
          "logType": "SAP_NETWEAVER_DEV_TRACE_LOGS",
          "monitor": true,
          "encoding": "utf-8"
        }
      ]
    }
  ],
  "netWeaverPrometheusExporter": {
    "sapSid": "ML4",
    "instanceNumbers": [
      "00",
      "11"
    ],
    "prometheusPort": "9680"
  }
}
```

Amazon EC2 での SAP Netweaver High Availability

次の例は、Amazon EC2 での SAP Netweaver High Availability 用の JSON 形式のコンポーネント設定を示しています。

```
{
  "subComponents": [
```

```
    {
      "subComponentType": "AWS::EC2::Instance",
      "alarmMetrics": [
        {
          "alarmMetricName": "ha_cluster_corosync_ring_errors",
          "monitor": true
        },
        {
          "alarmMetricName": "ha_cluster_pacemaker_fail_count",
          "monitor": true
        },
        {
          "alarmMetricName": "sap_HA_check_failover_config_state",
          "monitor": true
        },
        {
          "alarmMetricName": "sap_HA_get_failover_config_HAActive",
          "monitor": true
        },
        {
          "alarmMetricName": "sap_alerts_AbortedJobs",
          "monitor": true
        },
        {
          "alarmMetricName": "sap_alerts_Availability",
          "monitor": true
        },
        {
          "alarmMetricName": "sap_alerts_BasisSystem",
          "monitor": true
        },
        {
          "alarmMetricName": "sap_alerts_DBRequestTime",
          "monitor": true
        },
        {
          "alarmMetricName": "sap_alerts_Database",
          "monitor": true
        },
        {
          "alarmMetricName": "sap_alerts_FrontendResponseTime",
          "monitor": true
        },
        {
```

```
          "alarmMetricName": "sap_alerts_LongRunners",
          "monitor": true
        },
        {
          "alarmMetricName": "sap_alerts_QueueTime",
          "monitor": true
        },
        {
          "alarmMetricName": "sap_alerts_ResponseTime",
          "monitor": true
        },
        {
          "alarmMetricName": "sap_alerts_ResponseTimeDialog",
          "monitor": true
        },
        {
          "alarmMetricName": "sap_alerts_ResponseTimeDialogRFC",
          "monitor": true
        },
        {
          "alarmMetricName": "sap_alerts_Security",
          "monitor": true
        },
        {
          "alarmMetricName": "sap_alerts_Shortdumps",
          "monitor": true
        },
        {
          "alarmMetricName": "sap_alerts_SqlError",
          "monitor": true
        },
        {
          "alarmMetricName": "sap_alerts_System",
          "monitor": true
        },
        {
          "alarmMetricName": "sap_enqueue_server_replication_state",
          "monitor": true
        },
        {
          "alarmMetricName": "sap_start_service_processes",
          "monitor": true
        }
      ],
```

```
      "logs": [
        {
          "logGroupName": "SAP_NETWEAVER_DEV_TRACE_LOGS-NetWeaver-PR1",
          "logPath": "/usr/sap/<SID>/D*/work/dev_w*",
          "logType": "SAP_NETWEAVER_DEV_TRACE_LOGS",
          "monitor": true,
          "encoding": "utf-8"
        }
      ]
    }
  ],
  "haClusterPrometheusExporter": {
    "prometheusPort": "9664"
  },
  "netWeaverPrometheusExporter": {
    "sapSid": "PR1",
    "instanceNumbers": [
      "11",
      "12"
    ],
    "prometheusPort": "9680"
  }
}
```

SQL Always On 可用性グループ

次の例は、SQL Always On 可用性グループ用の JSON 形式のコンポーネント設定を示しています。

```
{
  "subComponents" : [ {
    "subComponentType" : "AWS::EC2::Instance",
    "alarmMetrics" : [ {
      "alarmMetricName" : "CPUUtilization",
      "monitor" : true
    }, {
      "alarmMetricName" : "StatusCheckFailed",
      "monitor" : true
    }, {
      "alarmMetricName" : "Processor % Processor Time",
      "monitor" : true
    }, {
      "alarmMetricName" : "Memory % Committed Bytes In Use",
      "monitor" : true
    }, {
```

```
      "alarmMetricName" : "Memory Available Mbytes",
      "monitor" : true
    }, {
      "alarmMetricName" : "Paging File % Usage",
      "monitor" : true
    }, {
      "alarmMetricName" : "System Processor Queue Length",
      "monitor" : true
    }, {
      "alarmMetricName" : "Network Interface Bytes Total/sec",
      "monitor" : true
    }, {
      "alarmMetricName" : "PhysicalDisk % Disk Time",
      "monitor" : true
    }, {
      "alarmMetricName" : "SQLServer:Buffer Manager Buffer cache hit ratio",
      "monitor" : true
    }, {
      "alarmMetricName" : "SQLServer:Buffer Manager Page life expectancy",
      "monitor" : true
    }, {
      "alarmMetricName" : "SQLServer:General Statistics Processes blocked",
      "monitor" : true
    }, {
      "alarmMetricName" : "SQLServer:General Statistics User Connections",
      "monitor" : true
    }, {
      "alarmMetricName" : "SQLServer:Locks Number of Deadlocks/sec",
      "monitor" : true
    }, {
      "alarmMetricName" : "SQLServer:SQL Statistics Batch Requests/sec",
      "monitor" : true
    }, {
      "alarmMetricName" : "SQLServer:Database Replica File Bytes Received/sec",
      "monitor" : true
    }, {
      "alarmMetricName" : "SQLServer:Database Replica Log Bytes Received/sec",
      "monitor" : true
    }, {
      "alarmMetricName" : "SQLServer:Database Replica Log remaining for undo",
      "monitor" : true
    }, {
      "alarmMetricName" : "SQLServer:Database Replica Log Send Queue",
      "monitor" : true
```

```
    }, {
      "alarmMetricName" : "SQLServer:Database Replica Mirrored Write Transaction/sec",
      "monitor" : true
    }, {
      "alarmMetricName" : "SQLServer:Database Replica Recovery Queue",
      "monitor" : true
    }, {
      "alarmMetricName" : "SQLServer:Database Replica Redo Bytes Remaining",
      "monitor" : true
    }, {
      "alarmMetricName" : "SQLServer:Database Replica Redone Bytes/sec",
      "monitor" : true
    }, {
      "alarmMetricName" : "SQLServer:Database Replica Total Log requiring undo",
      "monitor" : true
    }, {
      "alarmMetricName" : "SQLServer:Database Replica Transaction Delay",
      "monitor" : true
    } ],
    "windowsEvents" : [ {
      "logGroupName" : "WINDOWS_EVENTS-Application-<RESOURCE_GROUP_NAME>",
      "eventName" : "Application",
      "eventLevels" : [ "WARNING", "ERROR", "CRITICAL", "INFORMATION" ],
      "monitor" : true
    }, {
      "logGroupName" : "WINDOWS_EVENTS-System-<RESOURCE_GROUP_NAME>",
      "eventName" : "System",
      "eventLevels" : [ "WARNING", "ERROR", "CRITICAL" ],
      "monitor" : true
    }, {
      "logGroupName" : "WINDOWS_EVENTS-Security-<RESOURCE_GROUP_NAME>",
      "eventName" : "Security",
      "eventLevels" : [ "WARNING", "ERROR", "CRITICAL" ],
      "monitor" : true
    } ],
    "logs" : [ {
      "logGroupName" : "SQL_SERVER_ALWAYSON_AVAILABILITY_GROUP-<RESOURCE_GROUP_NAME>",
      "logPath" : "C:\\Program Files\\Microsoft SQL Server\\MSSQL**.MSSQLSERVER\\MSSQL\
\Log\\ERRORLOG",
      "logType" : "SQL_SERVER",
      "monitor" : true,
      "encoding" : "utf-8"
    } ]
  }, {
```

```
    "subComponentType" : "AWS::EC2::Volume",
    "alarmMetrics" : [ {
      "alarmMetricName" : "VolumeReadBytes",
      "monitor" : true
    }, {
    "alarmMetricName" : "VolumeWriteBytes",
      "monitor" : true
    }, {
    "alarmMetricName" : "VolumeReadOps",
      "monitor" : true
    }, {
    "alarmMetricName" : "VolumeWriteOps",
      "monitor" : true
    }, {
    "alarmMetricName" : "VolumeQueueLength",
      "monitor" : true
    }, {
    "alarmMetricName" : "VolumeThroughputPercentage",
      "monitor" : true
    }, {
    "alarmMetricName" : "BurstBalance",
      "monitor" : true
    } ]
  } ]
}
```

SQL フェイルオーバークラスターインスタンス

次の例は、SQL フェールオーバークラスターインスタンス用の JSON 形式のコンポーネント設定を示しています。

```
{
  "subComponents" : [ {
    "subComponentType" : "AWS::EC2::Instance",
    "alarmMetrics" : [ {
      "alarmMetricName" : "CPUUtilization",
      "monitor" : true
    }, {
      "alarmMetricName" : "StatusCheckFailed",
      "monitor" : true
    }, {
      "alarmMetricName" : "Processor % Processor Time",
      "monitor" : true
```

```
    }, {
      "alarmMetricName" : "Memory % Committed Bytes In Use",
      "monitor" : true
    }, {
      "alarmMetricName" : "Memory Available Mbytes",
      "monitor" : true
    }, {
      "alarmMetricName" : "Paging File % Usage",
      "monitor" : true
    }, {
      "alarmMetricName" : "System Processor Queue Length",
      "monitor" : true
    }, {
      "alarmMetricName" : "Network Interface Bytes Total/sec",
      "monitor" : true
    }, {
      "alarmMetricName" : "PhysicalDisk % Disk Time",
      "monitor" : true
    }, {
      "alarmMetricName" : "Bytes Received/sec",
      "monitor" : true
    }, {
      "alarmMetricName" : "Normal Messages Queue Length/sec",
      "monitor" : true
    }, {
      "alarmMetricName" : "Urgent Message Queue Length/se",
      "monitor" : true
    }, {
      "alarmMetricName" : "Reconnect Count",
      "monitor" : true
    }, {
      "alarmMetricName" : "Unacknowledged Message Queue Length/sec",
      "monitor" : true
    }, {
      "alarmMetricName" : "Messages Outstanding",
      "monitor" : true
    }, {
      "alarmMetricName" : "Messages Sent/sec",
      "monitor" : true
    }, {
      "alarmMetricName" : "Database Update Messages/sec",
      "monitor" : true
    }, {
      "alarmMetricName" : "Update Messages/sec",
```

```
      "monitor" : true
    }, {
      "alarmMetricName" : "Flushes/sec",
      "monitor" : true
    }, {
      "alarmMetricName" : "Crypto Checkpoints Saved/sec",
      "monitor" : true
    }, {
      "alarmMetricName" : "Crypto Checkpoints Restored/sec",
      "monitor" : true
    }, {
      "alarmMetricName" : "Registry Checkpoints Restored/sec",
      "monitor" : true
    }, {
      "alarmMetricName" : "Registry Checkpoints Saved/sec",
      "monitor" : true
    }, {
      "alarmMetricName" : "Cluster API Calls/sec",
      "monitor" : true
    }, {
      "alarmMetricName" : "Resource API Calls/sec",
      "monitor" : true
    }, {
      "alarmMetricName" : "Cluster Handles/sec",
      "monitor" : true
    }, {
      "alarmMetricName" : "Resource Handles/sec",
      "monitor" : true
    } ],
    "windowsEvents" : [ {
      "logGroupName" : "WINDOWS_EVENTS-Application-<RESOURCE_GROUP_NAME>",
      "eventName" : "Application",
      "eventLevels" : [ "WARNING", "ERROR", "CRITICAL"],
      "monitor" : true
    }, {
      "logGroupName" : "WINDOWS_EVENTS-System-<RESOURCE_GROUP_NAME>",
      "eventName" : "System",
      "eventLevels" : [ "WARNING", "ERROR", "CRITICAL", "INFORMATION" ],
      "monitor" : true
    }, {
      "logGroupName" : "WINDOWS_EVENTS-Security-<RESOURCE_GROUP_NAME>",
      "eventName" : "Security",
      "eventLevels" : [ "WARNING", "ERROR", "CRITICAL" ],
      "monitor" : true
```

```
    } ],
    "logs" : [ {
      "logGroupName" : "SQL_SERVER_FAILOVER_CLUSTER_INSTANCE-<RESOURCE_GROUP_NAME>",
      "logPath" : "\\\\amznfsxjmzbykwn.mydomain.aws\\SQLDB\\MSSQL**.MSSQLSERVER\\MSSQL\
\Log\\ERRORLOG",
      "logType" : "SQL_SERVER",
      "monitor" : true,
      "encoding" : "utf-8"
    } ]
  }, {
    "subComponentType" : "AWS::EC2::Volume",
    "alarmMetrics" : [ {
      "alarmMetricName" : "VolumeReadBytes",
      "monitor" : true
    }, {
    "alarmMetricName" : "VolumeWriteBytes",
      "monitor" : true
    }, {
    "alarmMetricName" : "VolumeReadOps",
      "monitor" : true
    }, {
    "alarmMetricName" : "VolumeWriteOps",
      "monitor" : true
    }, {
    "alarmMetricName" : "VolumeQueueLength",
      "monitor" : true
    }, {
    "alarmMetricName" : "VolumeThroughputPercentage",
      "monitor" : true
    }, {
    "alarmMetricName" : "BurstBalance",
      "monitor" : true
    } ]
  } ]
}
```

CloudFormation テンプレートを使用して CloudWatch Application Insights モニターリングを作成および設定する

AWS CloudFormation テンプレートから、主要メトリクスやテレメトリを含む Application Insights モニターリングをアプリケーション、データベース、ウェブサーバーに追加できます。

このセクションでは、Application Insights モニターリングの作成および設定に役立つサンプル AWS CloudFormation テンプレートを JSON 形式と YAML 形式の両方で提供しています。

AWS CloudFormation ユーザーガイドの Application Insights リソースとプロパティのリファレンスを表示するには、ApplicationInsights リソースタイプのリファレンスを参照してください。

サンプルテンプレート

- AWS CloudFormation スタック全体の Application Insights アプリケーションを作成する
- 詳細な設定を使用した Application Insights アプリケーションの作成
- CUSTOM モードのコンポーネント設定を使用した Application Insights アプリケーションの作成
- DEFAULT モードのコンポーネント設定を使用した Application Insights アプリケーションの作成
- DEFAULT_WITH_OVERWRITE モードのコンポーネント設定を使用した Application Insights アプリケーションの作成

AWS CloudFormation スタック全体の Application Insights アプリケーションを作成する

次のテンプレートを適用するには、AWS リソースと、それらのリソースをモニターリングする Application Insights アプリケーションの作成元となる 1 つ以上の Resource Groups を作成する必要があります。詳細については、「AWS Resource Groups の開始方法」を参照してください。

次のテンプレートの最初の 2 つの部分は、リソースとリソースグループを指定します。テンプレートの最後の部分では、リソースグループの Application Insights アプリケーションを作成しますが、アプリケーションの設定やモニターリングの適用は行いません。詳細については、Amazon CloudWatch Application Insights API リファレンスの CreateApplication コマンドの詳細を参照してください。

JSON 形式のテンプレート

```
{
    "AWSTemplateFormatVersion": "2010-09-09",
    "Description": "Test Resource Group stack",
    "Resources": {
        "EC2Instance": {
            "Type": "AWS::EC2::Instance",
            "Properties": {
                "ImageId" : "ami-abcd1234efgh5678i",
```

```
                "SecurityGroupIds" : ["sg-abcd1234"]
            }
        },
        ...
        "ResourceGroup": {
            "Type": "AWS::ResourceGroups::Group",
            "Properties": {
                "Name": "my_resource_group"
            }
        },
        "AppInsightsApp": {
            "Type": "AWS::ApplicationInsights::Application",
            "Properties": {
                "ResourceGroupName": "my_resource_group"
            },
            "DependsOn" : "ResourceGroup"
        }
    }
}
```

YAML 形式のテンプレート

```
---
AWSTemplateFormatVersion: '2010-09-09'
Description: Test Resource Group stack
Resources:
  EC2Instance:
    Type: AWS::EC2::Instance
    Properties:
      ImageId: ami-abcd1234efgh5678i
      SecurityGroupIds:
      - sg-abcd1234
  ...
  ResourceGroup:
    Type: AWS::ResourceGroups::Group
    Properties:
      Name: my_resource_group
  AppInsightsApp:
    Type: AWS::ApplicationInsights::Application
    Properties:
      ResourceGroupName: my_resource_group
    DependsOn: ResourceGroup
```

次のテンプレートセクションでは、デフォルトのモニターリング設定を Application Insights アプリケーションに適用します。詳細については、Amazon CloudWatch Application Insights API リファレンスの CreateApplication コマンドの詳細を参照してください。

AutoConfigurationEnabled を true に設定すると、アプリケーションのすべてのコンポーネントが、DEFAULT アプリケーション層の推奨モニターリング設定を使用して設定されます。これらの設定と層の詳細については、Amazon CloudWatch Application Insights API リファレンスの「DescribeComponentConfigurationRecommendation」と「UpdateComponentConfiguration」を参照してください。

JSON 形式のテンプレート

```
{
    "AWSTemplateFormatVersion": "2010-09-09",
    "Description": "Test Application Insights Application stack",
    "Resources": {
        "AppInsightsApp": {
            "Type": "AWS::ApplicationInsights::Application",
            "Properties": {
                "ResourceGroupName": "my_resource_group",
                "AutoConfigurationEnabled": true
            }
        }
    }
}
```

YAML 形式のテンプレート

```
---
AWSTemplateFormatVersion: '2010-09-09'
Description: Test Application Insights Application stack
Resources:
  AppInsightsApp:
    Type: AWS::ApplicationInsights::Application
    Properties:
      ResourceGroupName: my_resource_group
      AutoConfigurationEnabled: true
```

詳細な設定を使用した Application Insights アプリケーションの作成

次のテンプレートは、これらのアクションを実行します。

- CloudWatch Events 通知と OpsCenter が有効な Application Insights アプリケーションを作成します。詳細については、Amazon CloudWatch Application Insights API リファレンスの CreateApplication コマンドの詳細を参照してください。
- アプリケーションに 2 つのタグを付けます。そのうちの 1 つにはタグ値がありません。詳細については、Amazon CloudWatch Application Insights API リファレンスの「TagResource」を参照してください。
- 2 つのカスタムインスタンスグループコンポーネントを作成します。詳細については、Amazon CloudWatch Application Insights API リファレンスの「CreateComponent」を参照してください。
- 2 つのログパターンセットを作成します。詳細については、Amazon CloudWatch Application Insights API リファレンスの「CreateLogPattern」を参照してください。
- `AutoConfigurationEnabled` を `true` に設定します。これにより、アプリケーションのすべてのコンポーネントが、`DEFAULT` 層で推奨されるモニターリング設定で構成されます。詳細については、Amazon CloudWatch Application Insights API リファレンスの「DescribeComponentConfigurationRecommendation」を参照してください。

JSON 形式のテンプレート

```
{
    "Type": "AWS::ApplicationInsights::Application",
    "Properties": {
        "ResourceGroupName": "my_resource_group",
        "CWEMonitorEnabled": true,
        "OpsCenterEnabled": true,
        "OpsItemSNSTopicArn": "arn:aws:sns:us-east-1:123456789012:my_topic",
        "AutoConfigurationEnabled": true,
        "Tags": [
            {
                "Key": "key1",
                "Value": "value1"
            },
            {
                "Key": "key2",
                "Value": ""
            }
        ],
        "CustomComponents": [
            {
                "ComponentName": "test_component_1",
                "ResourceList": [
```

```
                    "arn:aws:ec2:us-east-1:123456789012:instance/i-abcd1234efgh5678i"
                ]
            },
            {
                "ComponentName": "test_component_2",
                "ResourceList": [
                    "arn:aws:ec2:us-east-1:123456789012:instance/i-abcd1234efgh5678i",
                    "arn:aws:ec2:us-east-1:123456789012:instance/i-abcd1234efgh5678i"
                ]
            }
        ],
        "LogPatternSets": [
            {
                "PatternSetName": "pattern_set_1",
                "LogPatterns": [
                    {
                        "PatternName": "deadlock_pattern",
                        "Pattern": ".*\\sDeadlocked\\sSchedulers(([^\\w].*)|($))",
                        "Rank": 1
                    }
                ]
            },
            {
                "PatternSetName": "pattern_set_2",
                "LogPatterns": [
                    {
                        "PatternName": "error_pattern",
                        "Pattern": ".*[\\s\\[]ERROR[\\s\\]].*",
                        "Rank": 1
                    },
                    {
                        "PatternName": "warning_pattern",
                        "Pattern": ".*[\\s\\[]WARN(ING)?[\\s\\]].*",
                        "Rank": 10
                    }
                ]
            }
        ]
    }
}
```

YAML 形式のテンプレート

```
---
Type: AWS::ApplicationInsights::Application
Properties:
  ResourceGroupName: my_resource_group
  CWEMonitorEnabled: true
  OpsCenterEnabled: true
  OpsItemSNSTopicArn: arn:aws:sns:us-east-1:123456789012:my_topic
  AutoConfigurationEnabled: true
  Tags:
  - Key: key1
    Value: value1
  - Key: key2
    Value: ''
  CustomComponents:
  - ComponentName: test_component_1
    ResourceList:
    - arn:aws:ec2:us-east-1:123456789012:instance/i-abcd1234efgh5678i
  - ComponentName: test_component_2
    ResourceList:
    - arn:aws:ec2:us-east-1:123456789012:instance/i-abcd1234efgh5678i
    - arn:aws:ec2:us-east-1:123456789012:instance/i-abcd1234efgh5678i
  LogPatternSets:
  - PatternSetName: pattern_set_1
    LogPatterns:
    - PatternName: deadlock_pattern
      Pattern: ".*\\sDeadlocked\\sSchedulers(([^\\w].*)|($))"
      Rank: 1
  - PatternSetName: pattern_set_2
    LogPatterns:
    - PatternName: error_pattern
      Pattern: ".*[\\s\\[]ERROR[\\s\\]].*"
      Rank: 1
    - PatternName: warning_pattern
      Pattern: ".*[\\s\\[]WARN(ING)?[\\s\\]].*"
      Rank: 10
```

CUSTOM モードのコンポーネント設定を使用した Application Insights アプリケーションの作成

次のテンプレートは、これらのアクションを実行します。

- Application Insights アプリケーションを作成します。詳細については、Amazon CloudWatch Application Insights API リファレンスの「CreateApplication」を参照してください。
- Component my_component は ComponentConfigurationMode を CUSTOM に設定します。これにより、このコンポーネントは CustomComponentConfiguration で指定された設定で設定されます。詳細については、Amazon CloudWatch Application Insights API リファレンスの「UpdateComponentConfiguration」を参照してください。

JSON 形式のテンプレート

```
{
    "Type": "AWS::ApplicationInsights::Application",
    "Properties": {
        "ResourceGroupName": "my_resource_group,
        "ComponentMonitoringSettings": [
            {
                "ComponentARN": "my_component",
                "Tier": "SQL_SERVER",
                "ComponentConfigurationMode": "CUSTOM",
                "CustomComponentConfiguration": {
                    "ConfigurationDetails": {
                        "AlarmMetrics": [
                            {
                                "AlarmMetricName": "StatusCheckFailed"
                            },
                            ...
                        ],
                        "Logs": [
                            {
                                "LogGroupName": "my_log_group_1",
                                "LogPath": "C:\\LogFolder_1\\*",
                                "LogType": "DOT_NET_CORE",
                                "Encoding": "utf-8",
                                "PatternSet": "my_pattern_set_1"
                            },
                            ...
                        ],
                        "WindowsEvents": [
                            {
                                "LogGroupName": "my_windows_event_log_group_1",
                                "EventName": "Application",
                                "EventLevels": [
                                    "ERROR",
```

```
                            "WARNING",
                            ...
                        ],
                        "Encoding": "utf-8",
                        "PatternSet": "my_pattern_set_2"
                    },
                    ...
                ],
                "Alarms": [
                    {
                        "AlarmName": "my_alarm_name",
                        "Severity": "HIGH"
                    },
                    ...
                ]
            },
            "SubComponentTypeConfigurations": [
                {
                    "SubComponentType": "EC2_INSTANCE",
                    "SubComponentConfigurationDetails": {
                        "AlarmMetrics": [
                            {
                                "AlarmMetricName": "DiskReadOps"
                            },
                            ...
                        ],
                        "Logs": [
                            {
                                "LogGroupName": "my_log_group_2",
                                "LogPath": "C:\\LogFolder_2\\*",
                                "LogType": "IIS",
                                "Encoding": "utf-8",
                                "PatternSet": "my_pattern_set_3"
                            },
                            ...
                        ],
                        "processes" : [
                            {
                                "processName" : "my_process",
                                "alarmMetrics" : [
                            {
                                "alarmMetricName" : "procstat cpu_usage",
                                "monitor" : true
                            }, {
```

```
                                    "alarmMetricName" : "procstat memory_rss",
                                    "monitor" : true
                                }
                            ]
                        }
                    ],
                            "WindowsEvents": [
                                {
                                    "LogGroupName": "my_windows_event_log_group_2",
                                    "EventName": "Application",
                                    "EventLevels": [
                                        "ERROR",
                                        "WARNING",
                                        ...
                                    ],
                                    "Encoding": "utf-8",
                                    "PatternSet": "my_pattern_set_4"
                                },
                                ...
                            ]
                        }
                    }
                ]
            }
        }
    ]
  }
}
```

YAML 形式のテンプレート

```
---
Type: AWS::ApplicationInsights::Application
Properties:
  ResourceGroupName: my_resource_group
  ComponentMonitoringSettings:
  - ComponentARN: my_component
    Tier: SQL_SERVER
    ComponentConfigurationMode: CUSTOM
    CustomComponentConfiguration:
      ConfigurationDetails:
        AlarmMetrics:
        - AlarmMetricName: StatusCheckFailed
```

```
        ...
        Logs:
        - LogGroupName: my_log_group_1
          LogPath: C:\LogFolder_1\*
          LogType: DOT_NET_CORE
          Encoding: utf-8
          PatternSet: my_pattern_set_1
        ...
        WindowsEvents:
        - LogGroupName: my_windows_event_log_group_1
          EventName: Application
          EventLevels:
          - ERROR
          - WARNING
          ...
          Encoding: utf-8
          PatternSet: my_pattern_set_2
        ...
        Alarms:
        - AlarmName: my_alarm_name
          Severity: HIGH
        ...
      SubComponentTypeConfigurations:
      - SubComponentType: EC2_INSTANCE
        SubComponentConfigurationDetails:
          AlarmMetrics:
          - AlarmMetricName: DiskReadOps
          ...
          Logs:
          - LogGroupName: my_log_group_2
            LogPath: C:\LogFolder_2\*
            LogType: IIS
            Encoding: utf-8
            PatternSet: my_pattern_set_3
          ...
          Processes:
          - ProcessName: my_process
            AlarmMetrics:
            - AlarmMetricName: procstat cpu_usage
              ...
            ...
          WindowsEvents:
          - LogGroupName: my_windows_event_log_group_2
            EventName: Application
```

```
            EventLevels:
            - ERROR
            - WARNING
            ...
            Encoding: utf-8
            PatternSet: my_pattern_set_4
          ...
```

DEFAULT モードのコンポーネント設定を使用した Application Insights アプリケーションの作成

次のテンプレートは、これらのアクションを実行します。

- Application Insights アプリケーションを作成します。詳細については、Amazon CloudWatch Application Insights API リファレンスの「CreateApplication」を参照してください。
- コンポーネント my_component は、ComponentConfigurationMode を DEFAULT に設定し、Tier を SQL_SERVER に設定します。これにより、このコンポーネントは、Application Insights が SQL_Server 層の推奨される構成設定を使用して構成されます。詳細については、Amazon CloudWatch Application Insights API リファレンスの「DescribeComponentConfiguration」と「UpdateComponentConfiguration」を参照してください。

JSON 形式のテンプレート

```
{
    "Type": "AWS::ApplicationInsights::Application",
    "Properties": {
        "ResourceGroupName": "my_resource_group",
        "ComponentMonitoringSettings": [
            {
                "ComponentARN": "my_component",
                "Tier": "SQL_SERVER",
                "ComponentConfigurationMode": "DEFAULT"
            }
        ]
    }
}
```

YAML 形式のテンプレート

```
---
Type: AWS::ApplicationInsights::Application
Properties:
  ResourceGroupName: my_resource_group
  ComponentMonitoringSettings:
  - ComponentARN: my_component
    Tier: SQL_SERVER
    ComponentConfigurationMode: DEFAULT
```

DEFAULT_WITH_OVERWRITE モードのコンポーネント設定を使用した Application Insights アプリケーションの作成

次のテンプレートは、これらのアクションを実行します。

- Application Insights アプリケーションを作成します。詳細については、Amazon CloudWatch Application Insights API リファレンスの「CreateApplication」を参照してください。
- コンポーネント my_component は、ComponentConfigurationMode を DEFAULT_WITH_OVERWRITE に設定し、tier を DOT_NET_CORE に設定します。これにより、このコンポーネントは、Application Insights が DOT_NET_CORE 層の推奨される構成設定を使用して構成されます。上書きする構成設定は、DefaultOverwriteComponentConfiguration で指定します。
  - コンポーネントレベルでは、AlarmMetrics 設定が上書きされます。
  - サブコンポーネントレベルでは、EC2_Instance タイプのサブコンポーネントで Logs 設定が上書きされます。

  詳細については、Amazon CloudWatch Application Insights API リファレンスの「UpdateComponentConfiguration」を参照してください。

JSON 形式のテンプレート

```
{
    "Type": "AWS::ApplicationInsights::Application",
    "Properties": {
        "ResourceGroupName": "my_resource_group",
        "ComponentMonitoringSettings": [
            {
                "ComponentName": "my_component",
                "Tier": "DOT_NET_CORE",
                "ComponentConfigurationMode": "DEFAULT_WITH_OVERWRITE",
```

```
                "DefaultOverwriteComponentConfiguration": {
                    "ConfigurationDetails": {
                        "AlarmMetrics": [
                            {
                                "AlarmMetricName": "StatusCheckFailed"
                            }
                        ]
                    },
                    "SubComponentTypeConfigurations": [
                        {
                            "SubComponentType": "EC2_INSTANCE",
                            "SubComponentConfigurationDetails": {
                                "Logs": [
                                    {
                                        "LogGroupName": "my_log_group",
                                        "LogPath": "C:\\LogFolder\\*",
                                        "LogType": "IIS",
                                        "Encoding": "utf-8",
                                        "PatternSet": "my_pattern_set"
                                    }
                                ]
                            }
                        }
                    ]
                }
            }
        ]
    }
}
```

YAML 形式のテンプレート

```
---
Type: AWS::ApplicationInsights::Application
Properties:
  ResourceGroupName: my_resource_group
  ComponentMonitoringSettings:
  - ComponentName: my_component
    Tier: DOT_NET_CORE
    ComponentConfigurationMode: DEFAULT_WITH_OVERWRITE
    DefaultOverwriteComponentConfiguration:
      ConfigurationDetails:
        AlarmMetrics:
```

```
        - AlarmMetricName: StatusCheckFailed
      SubComponentTypeConfigurations:
      - SubComponentType: EC2_INSTANCE
        SubComponentConfigurationDetails:
          Logs:
          - LogGroupName: my_log_group
            LogPath: C:\LogFolder\*
            LogType: IIS
            Encoding: utf-8
            PatternSet: my_pattern_set
```

チュートリアル: SAP ASE のモニタリングをセットアップする

このチュートリアルでは、SAP ASE データベースのモニタリングをセットアップできるよう CloudWatch Application Insights を設定する方法を説明します。CloudWatch Application Insights の自動ダッシュボードを使用して、問題の詳細を可視化し、トラブルシューティングを加速させ、SAP ASE データベースでの解決までの平均時間 (MTTR) を短縮できます。

SAP ASE トピック用の Application Insights

- サポートされている環境
- サポートされるオペレーティングシステム
- 機能
- 前提条件
- SAP ASE データベースをモニタリング用にセットアップする
- SAP ASE データベースのモニタリングの管理
- アラームのしきい値を設定する
- Application Insights で検出された SAP ASE における問題の表示とトラブルシューティング
- SAP ASE 向けの Application Insights のトラブルシューティング

サポートされている環境

CloudWatch Application Insights では、次のシステムおよびパターン用に AWS リソースのデプロイがサポートされています。SAP ASE データベースソフトウェアおよびサポートされている SAP アプリケーションソフトウェアを指定してインストールします。

- 単一の Amazon EC2 インスタンスでの 1 つ以上の SAP ASE データベース — 単一ノードのスケールアップアーキテクチャの SAP ASE。

- クロス AZ SAP ASE データベースの高可用性セットアップ — SUSE/RHEL クラスタリングにより 2 つのアベイラビリティーゾーンでの高可用性が設定されている SAP ASE。

> **Note**
>
> CloudWatch Application Insights は、単一の SAP システム ID (SID) の ASE HA 環境のみをサポートしています。複数の ASE HA SID がアタッチされている場合、最初に検出された SID に対してのみモニタリングがセットアップされます。

サポートされるオペレーティングシステム

SAP ASE 向けの CloudWatch Application Insights は、次のオペレーティングシステム上で x86-64 アーキテクチャをサポートしています。

- SuSE Linux 12 SP4
- SuSE Linux 12 SP5
- SuSE Linux 15
- SuSE Linux 15 SP1
- SuSE Linux 15 SP2
- SuSE Linux 15 SP3
- SuSE Linux 15 SP4
- SuSE Linux 15 SP1 For SAP
- SuSE Linux 15 SP2 For SAP
- SuSE Linux 15 SP3 For SAP
- SuSE Linux 15 SP4 For SAP
- SuSE Linux 12 SP4 For SAP
- SuSE Linux 12 SP5 For SAP
- RedHat Linux 7.6
- RedHat Linux 7.7
- RedHat Linux 7.9
- RedHat Linux 8.1

- RedHat Linux 8.4
- RedHat Linux 8.6

機能

SAP ASE 向けの CloudWatch Application Insights には、次の機能があります。

- SAP ASE ワークロードの自動検出
- 静的しきい値に基づく SAP ASE アラームの自動作成
- 異常検出に基づく SAP ASE アラームの自動作成
- SAP ASE のログパターンの自動認識
- SAP ASE 用のヘルスダッシュボード
- SAP ASE 用の問題ダッシュボード

前提条件

CloudWatch Application Insights で SAP ASE データベースを設定するには、次の前提条件を実行する必要があります。

- SAP ASE 設定パラメータ — ASE データベースで、設定パラメータ "enable monitoring"、"sql text pipe max messages"、"sql text pipe active" を有効にする必要があります。これにより、CloudWatch Application Insights でデータベース用の完全なモニタリング機能が提供されます。ASE データベースでこれらの設定が有効になっていない場合、Application Insights ではモニタリングに必要なメトリクスを収集できるよう、これらが自動的に有効になります。
- SAP ASE データベースユーザー — Application Insights のオンボーディング時に提供されるデータベースユーザーには、次にアクセスするためのアクセス許可が必要です。
  - マスターデータベースとユーザー (テナント) データベースのシステムテーブル
  - モニタリングテーブル
- SAPHostCtrl — Amazon EC2 インスタンスで SAPHostCtrl をインストールおよびセットアップします。
- Amazon CloudWatch エージェント — Amazon EC2 インスタンスで既存の CloudWatch エージェントが実行されていないことを確認します。CloudWatch エージェントがインストールされている場合は、マージ競合を避けるために、既存の CloudWatch エージェント設定ファイルから CloudWatch Application Insights で使用しているリソースの設定を削除するようにしてください。

詳細については、「CloudWatch エージェント設定ファイルを手動で作成または編集する」を参照してください。

- AWS Systems Manager の有効化 — インスタンスに SSM Agent をインストールし、SSM に対して有効なインスタンスを有効にします。SSM Agent のインストールについての詳細は、AWS Systems Manager ユーザーガイドの「SSM Agent の使用」を参照してください。
- Amazon EC2 インスタンスロール — 次の Amazon EC2 インスタンスロールをアタッチして、データベースを設定する必要があります。
  - Systems Manager を有効にする `AmazonSSMManagedInstanceCore` ロールをアタッチする必要があります。詳細については、「AWS Systems Manager アイデンティティベースのポリシーの例」を参照してください。
  - インスタンスメトリクスとログが CloudWatch を介して発行されるように `CloudWatchAgentServerPolicy` をアタッチする必要があります。詳細については、「CloudWatch エージェントで使用する IAM ロールとユーザーを作成する」を参照してください。
  - AWS Secrets Manager に保存されているパスワードを読み取るには、次の IAM インラインポリシーを Amazon EC2 インスタンスロールにアタッチする必要があります。インラインポリシーの詳細については、AWS Identity and Access Management ユーザーガイドの「インラインポリシー」を参照してください。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": [
                "secretsmanager:GetSecretValue"
            ],
            "Resource": "arn:aws:secretsmanager:*:*:secret:ApplicationInsights-*"
        }
    ]
}
```

- AWS Resource Groups — アプリケーションを CloudWatch Application Insights にオンボードするには、アプリケーションスタックで使用されるすべての関連する AWS リソースを含むリソースグループを作成する必要があります。これには、SAP ASE データベースを実行する Amazon EC2 インスタンスと Amazon EBS ボリュームが含まれます。1 つのアカウントに複数のデータベースが

ある場合は、各 SAP ASE データベースシステムの AWS リソースを含む 1 つのリソースグループを作成することをお勧めします。

- IAM アクセス許可 — 管理者以外のユーザーの場合:
  - Application Insights でサービスにリンクされたロールを作成できる AWS Identity and Access Management (IAM) ポリシーを作成し、ユーザー ID にアタッチする必要があります。ポリシーをアタッチする手順については、「IAM ポリシー」を参照してください。
  - ユーザーには、AWS Secrets Manager でシークレットを作成し、データベースのユーザー認証情報を保存するためのアクセス許可が必要です。詳細については、「例: シークレット値を取得するアクセス許可」を参照してください。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "secretsmanager:CreateSecret"
      ],
      "Resource": "arn:aws:secretsmanager:*:*:secret:ApplicationInsights-*"
    }
  ]
}
```

- サービスにリンクされたロール – Application Insights は AWS Identity and Access Management (IAM) サービスにリンクされたロールを使用します。Application Insights コンソールで新しい Application Insights アプリケーションを作成する際、サービスにリンクされたロールが作成されます。詳細については、「CloudWatch Application Insights のサービスにリンクされたロールの使用」を参照してください。

SAP ASE データベースをモニタリング用にセットアップする

SAP ASE データベースのモニタリングをセットアップするには、次の手順を実行します。

1. CloudWatch コンソールを開きます。
2. 左側のナビゲーションペインで、[Insights] の下にある [Application Insights] を選択します。
3. [Application Insights] ページには、Application Insights でモニターリングするアプリケーションのリストと、各アプリケーションのモニターリングのステータスが表示されます。右上隅の [Add an application] (アプリケーションの追加) を選択します。

4. [アプリケーションの詳細を指定] ページで、[リソースグループ] のドロップダウンリストから SAP ASE データベースのリソースを含む AWS リソースグループを選択します。アプリケーションのリソースグループを作成していない場合は、[Resource group] (リソースグループ) のドロップダウンから、[Create new resource group] (新しいリソースグループの作成) を選択して作成できます。リソースグループの作成についての詳細は、AWS Resource Groups ユーザーガイドを参照してください。
5. [Monitor CloudWatch Events] (CloudWatch Events のモニタリング) で、CloudWatch Events と Application Insights のモニタリングを統合できるチェックボックスをオンにすると、Amazon EBS、Amazon EC2、AWS CodeDeploy、Amazon ECS、AWS Health APIs And Notifications、Amazon RDS、Amazon S3、および AWS Step Functions からのインサイトを取得できます。
6. 選択したアプリケーションの問題が検出されたときにそれらを表示し、通知を受け取るには、[AWS Systems Manager OpsCenter と統合する] にある [修正アクションのため AWS Systems Manager OpsCenter OpsItems を生成する] 横のチェックボックスをオンにします。AWS リソースに関連する OpsItems というオペレーション作業項目を解決するために実行されたオペレーションを追跡するには、SNS トピックの ARN を指定します。
7. オプションでタグを入力して、リソースを特定し整理できます。CloudWatch Application Insights では、タグベースのリソースグループと AWS CloudFormation スタックベースのリソースグループ (Application Auto Scaling グループを除く) の両方がサポートされています。詳細については、AWS Resource Groups とタグユーザーガイドの「Tag Editor」を参照してください。
8. [Next] (次へ) を選択してモニターリングの設定を続行します。
9. [検出されたコンポーネントを確認] ページには、CloudWatch Application Insights によって自動的に検出されたモニターリング対象のコンポーネントとそれらのワークロードが一覧表示されます。

   > **Note**
   >
   > 検出された SAP ASE High Availability ワークロードを含むコンポーネントは、1 つのコンポーネントで 1 つのワークロードのみをサポートします。検出された SAP ASE 単一ノードワークロードを含むコンポーネントは複数のワークロードをサポートしますが、ワークロードを追加したり削除することはできません。自動的に検出されたワークロードはすべてモニタリングされます。

10. [Next] を選択します。

11. [コンポーネントの詳細を指定] ページで、SAP ASE データベースのユーザー名とパスワードを入力します。
12. アプリケーションのモニターリング設定を確認し [Submit] (送信) をクリックします。
13. アプリケーション詳細ページが開き、[アプリケーションの概要] および [モニタリング対象コンポーネントとワークロード] と [モニタリング対象ではないコンポーネントとワークロード] のリストを表示できます。コンポーネントまたはワークロードの横にあるラジオボタンをオンにすると、[設定履歴]、[ログパターン] 、および作成したタグも表示できます。設定を送信すると、アカウントで SAP ASE システムのすべてのメトリクスとアラームがデプロイされます。これには最大 2 時間かかります。

SAP ASE データベースのモニタリングの管理

次の手順を実行して、SAP ASE データベースのユーザー認証情報、メトリクス、およびログパスを管理できます。

1. CloudWatch コンソールを開きます。
2. 左側のナビゲーションペインで、[Insights] の下にある [Application Insights] を選択します。
3. [Application Insights] ページには、Application Insights でモニターリングするアプリケーションのリストと、各アプリケーションのモニターリングのステータスが表示されます。
4. [Monitored components] (モニターリングされているコンポーネント) で、コンポーネント名の横にあるラジオボタンをオンにします。次に、[Manage monitoring] (モニターリングの管理) を選択します。
5. [EC2 instance group logs] (EC2 インスタンスグループのログ) では、既存のログパス、ログパターンのセット、ロググループ名を更新できます。さらに、追加で Application logs (アプリケーションログ) を最大 3 つまで追加できます。
6. [メトリクス] では、要件に応じて SAP ASE メトリクスを選択できます。SAP ASE のメトリクス名は、`asedb` で始まります。コンポーネントごとに最大 60 個のメトリクスを追加できます。
7. [ASE の設定] で、SAP ASE データベースのユーザー名とパスワードを入力します。これらは、Amazon CloudWatch エージェントが SAP ASE データベースに接続するために使用するユーザー名とパスワードです。
8. [Custom alarms] (カスタムアラーム) では、CloudWatch Application Insights によってモニターリングするアラームを追加できます。
9. アプリケーションのモニターリング設定を確認し、[Submit] (送信) をクリックします。設定を送信すると、アカウントで SAP HANA システムのすべてのメトリクスとアラームが更新されます。これには最大 2 時間かかります。

アラームのしきい値を設定する

CloudWatch Application Insights では、メトリクスのしきい値とともに監視するアラームの Amazon CloudWatch メトリクスが自動的に作成されます。指定した評価期間数にわたってメトリクスがしきい値を超えると、アラームは ALARM 状態に変わります。これらの設定は、Application Insights では保持されないことに注意してください。

単一のメトリクスのアラームを編集するには、次の手順を実行します。

1. CloudWatch コンソールを開きます。
2. ナビゲーションペインで、[Alarms] (アラーム)、[All alarms] (すべてのアラーム) の順に選択します。
3. CloudWatch Application Insights によって自動的に作成されたアラームの横にあるラジオボタンをオンにします。次に [Actions] (アクション) を選択し、ドロップダウンメニューから [Edit] (編集)を選択します。
4. [Metrics] (メトリクス) で次のパラメータを編集します。
   a. [Statistic] (統計) で、統計または事前定義済みのパーセンタイルのいずれかを選択するか、カスタムパーセンタイルを指定します。例えば、p95.45 と指定します。
   b. [Period] (期間) で、アラームの評価期間を選択します。アラームを評価する際、各期間は 1 つのデータポイントに集約されます。
5. [Conditions] (条件) で次のパラメータを編集します。
   a. メトリクスがしきい値より大きい、小さい、またはしきい値と等しいのいずれかを指定します。
   b. しきい値を指定します。
6. [Additional configuration] (追加設定) で、次のパラメータを編集します。
   a. [Datapoints to alarm] (アラームを実行するデータポイント) で、評価期間またはデータポイントの数を指定します。アラームを開始するため、ALARM 状態である必要があります。2 つの値が一致するとアラームが作成され、指定された連続する期間数を超過した場合、ALARM 状態に入ります。n 個中 m 個のアラームを作成するには、2 番目の値よりも小さい数字を最初の値に指定します。アラームの評価の詳細については、「アラームの評価」を参照してください。
   b. [Missing data treatment] (欠落データの処理) で、一部のデータポイントが欠落しているときのアラームの動作を選択します。欠落データの処理についての詳細は、「CloudWatch アラームの欠落データの処理の設定」を参照してください。

c. モニターリングする統計としてパーセンタイルをアラームで使用している場合は、[サンプル数が少ないパーセンタイル] ボックスが表示されます。サンプル数が少ないケースを評価するか無視するかを選択します。[無視 (アラーム状態を維持する)] を選択すると、サンプル数が少なすぎる場合でも現在のアラーム状態が常に維持されます。サンプルが少ないパーセンタイルの詳細については、「パーセンタイルベースの CloudWatch アラームおよび少数のデータサンプル」を参照してください。

7. [Next] を選択します。
8. [通知] で、アラームが ALARM 状態、OK 状態、または INSUFFICIENT_DATA 状態のときに通知するための SNS トピックを選択します。
9. [Update alarm] (アラームの更新) を選択します。

Application Insights で検出された SAP ASE における問題の表示とトラブルシューティング

このセクションは、Application Insights で SAP ASE のモニタリングを設定する際に発生する一般的なトラブルシューティングの問題を解決するのに役立ちます。

SAP ASE Backup Server エラー

エラーメッセージは、動的に作成されたダッシュボードを確認することで確認できます。ダッシュボードには、SAP ASE Backup Server で報告されたエラーメッセージが表示されます。SAP ASE Backup Server ログの詳細については、SAP ドキュメントの「Backup Server のエラーロギング」を参照してください。

SAP ASE のトランザクションの実行時間が長い

長時間実行されているトランザクションを特定し、停止できるのか、または実行時間が計画的なものかを確認します。詳細については、「2180410 - How to display transaction log records for long running transactions? - SAP ASE」を参照してください。

SAP ASE User 接続

SAP ASE データベースが、データベースで実行する予定のワークロードに適したサイズになっているかどうかを確認します。詳細については、SAP ドキュメントの「Configuring User Connections」を参照してください。

SAP ASE のディスク容量

動的に作成されるダッシュボードを確認することで、問題が起きているデータベース層を特定できます。ダッシュボードには、関連するメトリクスやログファイルのスニペットが表示されます。ディスク増大の原因を理解し、該当する場合は物理ディスクサイズ、割り当てられたディスク容量、またはその両方を増やすことが重要です。詳細については、SAP ドキュメントの「disk resize」を参照してください。

SAP ASE 向けの Application Insights のトラブルシューティング

このセクションでは、Application Insights ダッシュボードから返される一般的なエラーを解決するための手順について説明します。

エラー	返されたエラー	根本原因	解決方法
60 を超えるモニタリングメトリクスは追加できません。	`Component cannot have more than 60 monitored metric`	現在のメトリクスの上限は、コンポーネントごとに 60 個です。	制限を守るのに必要のないメトリクスは削除してください。
オンボーディングプロセスの後、SAP メトリクスやアラームは表示されません	AWS Systems Manager で `AWS-ConfigureAWSPackage` の run コマンドが失敗しました。出力では、次のエラーが表示されます: `CT-LIBRARY error:ct_connect(): protocol specific layer: external error: The attempt to connect to the server failed`	ユーザー名とパスワードが間違っている可能性があります。	ユーザー名とパスワードが有効であることを確認し、その後オンボーディングプロセスに戻ってください。

チュートリアル: SAP HANA のモニターリングを設定する

このチュートリアルでは、SAP HANA データベースのモニターリングを設定できるよう CloudWatch Application Insights を設定する方法を説明します。CloudWatch Application Insights の自動ダッシュボードを使用して、問題の詳細を可視化し、トラブルシューティングを加速させ、SAP HANA データベースでの解決までの平均時間 (MTTR) を短縮できます。

SAP HANA トピック用の Application Insights

- サポートされている環境
- サポートされるオペレーティングシステム
- 機能
- 前提条件
- SAP HANA データベースをモニターリング用に設定する
- SAP HANA データベースのモニターリングの管理
- CloudWatch Application Insights で検出された SAP HANA における問題の表示とトラブルシューティング
- SAP HANA の異常検出
- SAP HANA 向けの Application Insights のトラブルシューティング

サポートされている環境

CloudWatch Application Insights では、次のシステムおよびパターン用に AWS リソースのデプロイがサポートされています。SAP HANA データベースソフトウェアおよびサポートされている SAP アプリケーションソフトウェアを指定してインストールします。

- 単一の Amazon EC2 インスタンスでの SAP HANA データベース – シングルノードのスケールアップアーキテクチャにおける SAP HANA で、メモリは最大 24 TB です。
- 複数の Amazon EC2 インスタンスでの SAP HANA データベース – マルチノードのスケールアウトアーキテクチャにおける SAP HANA です。
- クロス AZ SAP HANA データベースの高可用性セットアップ – SUSE/RHEL クラスタリングにより 2 つのアベイラビリティーゾーンでの高可用性が設定されている SAP HANA。

> **Note**
>
> CloudWatch Application Insights は、単一の SID HANA 環境のみをサポートしています。複数の HANA SID がアタッチされている場合、最初に検出された SID に対してのみモニターリングがセットアップされます。

サポートされるオペレーティングシステム

SAP HANA 向けの CloudWatch Application Insights は、次のオペレーティングシステム上で x86-64 アーキテクチャをサポートしています。

- SuSE Linux 12 SP4 For SAP
- SuSE Linux 12 SP5 For SAP
- SuSE Linux 15
- SuSE Linux 15 SP1
- SuSE Linux 15 SP2
- SuSE Linux 15 For SAP
- SuSE Linux 15 SP1 For SAP
- SuSE Linux 15 SP2 For SAP
- SuSE Linux 15 SP3 For SAP
- SuSE Linux 15 SP4 For SAP
- SuSE Linux 15 SP5 For SAP
- RedHat Linux 8.6 For SAP With High Availability and Update Services
- RedHat Linux 8.5 For SAP With High Availability and Update Services
- RedHat Linux 8.4 For SAP With High Availability and Update Services
- RedHat Linux 8.3 For SAP With High Availability and Update Services
- RedHat Linux 8.2 For SAP With High Availability and Update Services
- RedHat Linux 8.1 For SAP With High Availability and Update Services
- RedHat Linux 7.9 For SAP With High Availability and Update Services

機能

SAP HANA 向けの CloudWatch Application Insights には、次の機能があります。

- SAP HANA ワークロードの自動検出
- 静的しきい値に基づく SAP HANA アラームの自動作成
- 異常検出に基づく SAP HANA アラームの自動作成
- SAP HANA のログパターンの自動認識
- SAP HANA 用のヘルスダッシュボード
- SAP HANA 用の問題ダッシュボード

前提条件

CloudWatch Application Insights で SAP HANA データベースを設定するには、次の前提条件を実行する必要があります。

- SAP HANA – 実行中およびアクセス可能な SAP HANA データベース 2.0 SPS05 を Amazon EC2 インスタンスにインストールします。
- SAP HANA データベースユーザー – モニタリングロールを持つデータベースユーザーを、SYSTEM データベースとすべてのテナントに作成する必要があります。

  例

  次の SQL コマンドを実行して、モニターリングロールを持つユーザーを作成します。

  ```
  su - <sid>adm
  hdbsql -u SYSTEM -p <SYSTEMDB password> -d SYSTEMDB
  CREATE USER CW_HANADB_EXPORTER_USER PASSWORD <Monitoring user password> NO
   FORCE_FIRST_PASSWORD_CHANGE;
  CREATE ROLE CW_HANADB_EXPORTER_ROLE;
  GRANT MONITORING TO CW_HANADB_EXPORTER_ROLE;
  GRANT CW_HANADB_EXPORTER_ROLE TO CW_HANADB_EXPORTER_USER;
  ```

- Python 3.8 – オペレーティングシステムに Python 3.8 以降のバージョンをインストールします。Python の最新リリース版を使用してください。オペレーティングシステムで Python3 が検出されない場合、Python 3.6 がインストールされます。

  詳細については、「installation example」を参照してください。

> **Note**
>
> SuSE Linux 15 SP4、RedHat Linux 8.6、およびそれ以降のオペレーティングシステムには Python 3.8 以降を手動でインストールする必要があります。

- Pip3 – インストーラプログラム pip3 をオペレーティングシステムにインストールします。オペレーティングシステムで pip3 が検出されない場合、インストールされます。
- hdbclient — CloudWatch Application Insights は、python ドライバーを使用して SAP HANA データベースに接続します。クライアントが python3 にインストールされていない場合は、hdbclient tar ファイルのバージョン `2.10 or later` が `/hana/shared/SID/hdbclient/` に存在することを確認してください。
- Amazon CloudWatch エージェント — Amazon EC2 インスタンスで既存の CloudWatch エージェントが実行されていないことを確認します。CloudWatch エージェントがインストールされている場合は、マージ競合を避けるために、既存の CloudWatch エージェント設定ファイルから CloudWatch Application Insights で使用しているリソースの設定を削除するようにしてください。詳細については、「CloudWatch エージェント設定ファイルを手動で作成または編集する」を参照してください。
- AWS Systems Manager の有効化 – インスタンスに SSM Agent をインストールし、インスタンスを SSM に対して有効にします。SSM Agent のインストールについての詳細は、「AWS Systems Manager ユーザーガイド」の「SSM Agent の使用」を参照してください。
- Amazon EC2 インスタンスロール — 次の Amazon EC2 インスタンスロールをアタッチして、データベースを設定する必要があります。
  - Systems Manager を有効にする `AmazonSSMManagedInstanceCore` ロールをアタッチする必要があります。詳細については、「AWS Systems Manager アイデンティティベースのポリシーの例」を参照してください。
  - インスタンスメトリクスとログが CloudWatch を介して発行されるように `CloudWatchAgentServerPolicy` をアタッチする必要があります。詳細については、「CloudWatch エージェントで使用する IAM ロールとユーザーを作成する」を参照してください
  - AWS Secrets Manager に保存されているパスワードを読み取るには、次の IAM インラインポリシーを Amazon EC2 インスタンスロールにアタッチする必要があります。インラインポリシーの詳細については、AWS Identity and Access Management ユーザーガイドの「インラインポリシー」を参照してください。

```
{
```

```
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": [
                "secretsmanager:GetSecretValue"
            ],
            "Resource": "arn:aws:secretsmanager:*:*:secret:ApplicationInsights-*"
        }
    ]
}
```

- AWS リソースグループ – アプリケーションを CloudWatch Application Insights にオンボードするには、アプリケーションスタックで使用されるすべての関連する AWS リソースを含むリソースグループを作成する必要があります。これには、SAP HANA データベースを実行する Amazon EC2 インスタンスと Amazon EBS ボリュームが含まれます。1 つのアカウントに複数のデータベースがある場合は、各 SAP HANA データベースシステムの AWS リソースを含む 1 つのリソースグループを作成することをお勧めします。
- IAM アクセス許可 — 管理者以外のユーザーの場合:
  - Application Insights でサービスにリンクされたロールを作成できる AWS Identity and Access Management (IAM) ポリシーを作成し、ユーザー ID にアタッチする必要があります。ポリシーをアタッチする手順については、「IAM ポリシー」を参照してください。
  - ユーザーには、AWS Secrets Manager でシークレットを作成し、データベースのユーザー認証情報を保存するためのアクセス許可が必要です。詳細については、「例: シークレット値を取得するアクセス許可」を参照してください。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "secretsmanager:CreateSecret"
      ],
      "Resource": "arn:aws:secretsmanager:*:*:secret:ApplicationInsights-*"
    }
  ]
}
```

- サービスにリンクされたロール – Application Insights は AWS Identity and Access Management (IAM) サービスにリンクされたロールを使用します。Application Insights コンソールで新しい Application Insights アプリケーションを作成する際、サービスにリンクされたロールが作成されます。詳細については、「CloudWatch Application Insights のサービスにリンクされたロールの使用」を参照してください。

SAP HANA データベースをモニターリング用に設定する

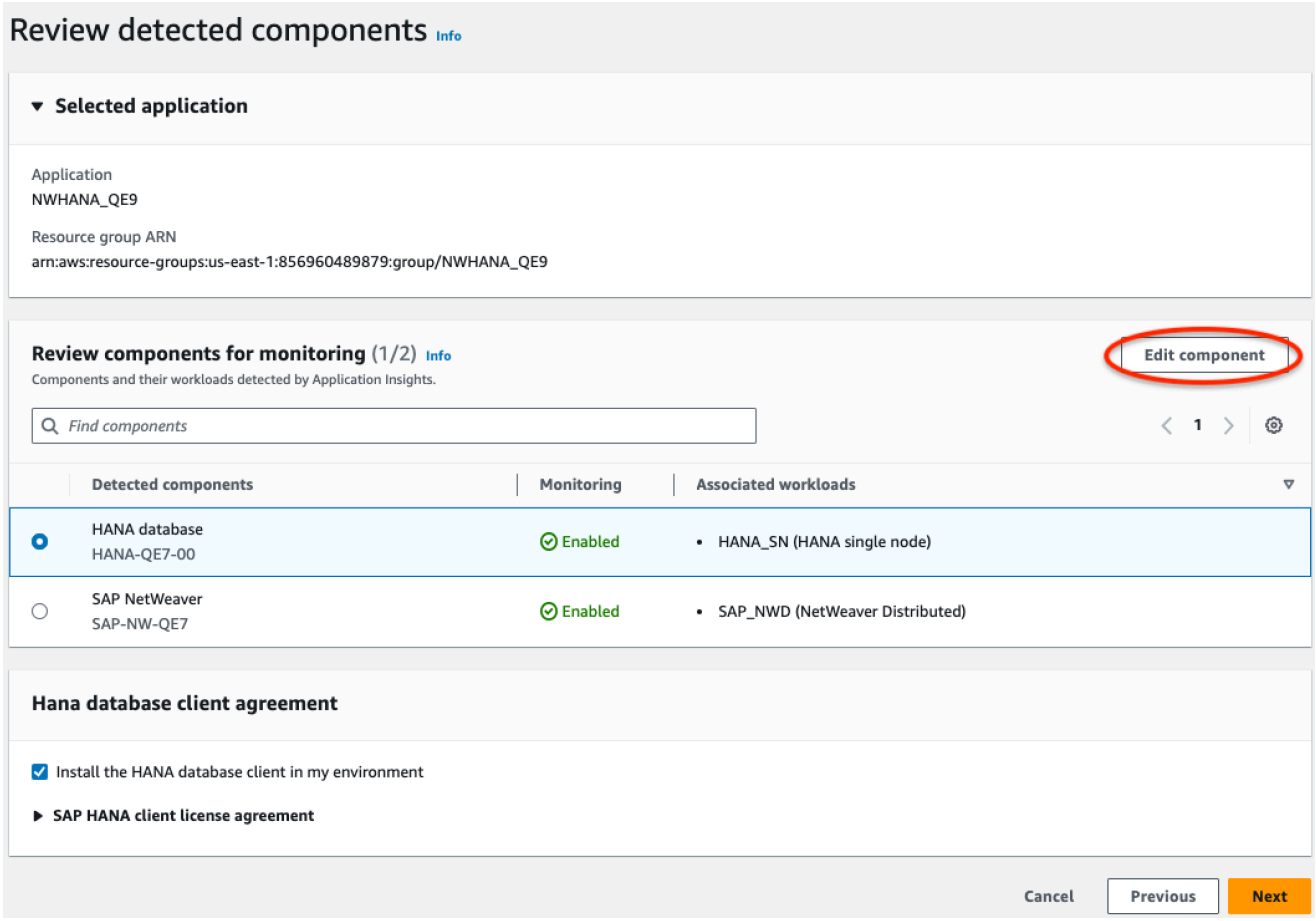
SAP HANA データベースのモニターリングをセットアップするには、以下の手順を実行します。

1. CloudWatch コンソールを開きます。
2. 左側のナビゲーションペインで、[Insights] の下にある [Application Insights] を選択します。
3. [Application Insights] ページには、Application Insights でモニターリングするアプリケーションのリストと、各アプリケーションのモニターリングのステータスが表示されます。右上隅の [Add an application] (アプリケーションの追加) を選択します。
4. [Specify application details] (アプリケーションの詳細の指定) ページで、[Resource group] (リソースグループ) のドロップダウンリストから SAP HANA データベースのリソースを含む AWS リソースグループを選択します。アプリケーションのリソースグループを作成していない場合は、[Resource group] (リソースグループ) のドロップダウンから、[Create new resource group] (新しいリソースグループの作成) を選択して作成できます。リソースグループの作成についての詳細は、AWS Resource Groups ユーザーガイドを参照してください。
5. [Monitor CloudWatch Events] (CloudWatch Events のモニタリング) で、CloudWatch Events と Application Insights のモニタリングを統合できるチェックボックスをオンにすると、Amazon EBS、Amazon EC2、AWS CodeDeploy、Amazon ECS、AWS Health APIs And Notifications、Amazon RDS、Amazon S3、および AWS Step Functions からのインサイトを取得できます。
6. 選択したアプリケーションの問題が検出されたときにそれらを表示し、通知を受け取るには、[AWS Systems Manager OpsCenter と統合する] にある [修正アクションのため AWS Systems Manager OpsCenter OpsItems を生成する] 横のチェックボックスをオンにします。AWS リソースに関連する OpsItems というオペレーション作業項目を解決するために実行されたオペレーションを追跡するには、SNS トピックの ARN を指定します。
7. オプションでタグを入力して、リソースを特定し整理できます。CloudWatch Application Insights では、タグベースのリソースグループと AWS CloudFormation スタックベースのリソースグループ (Application Auto Scaling グループを除く) の両方がサポートされています。詳細については、AWS Resource Groups とタグユーザーガイドの「Tag Editor」を参照してください。

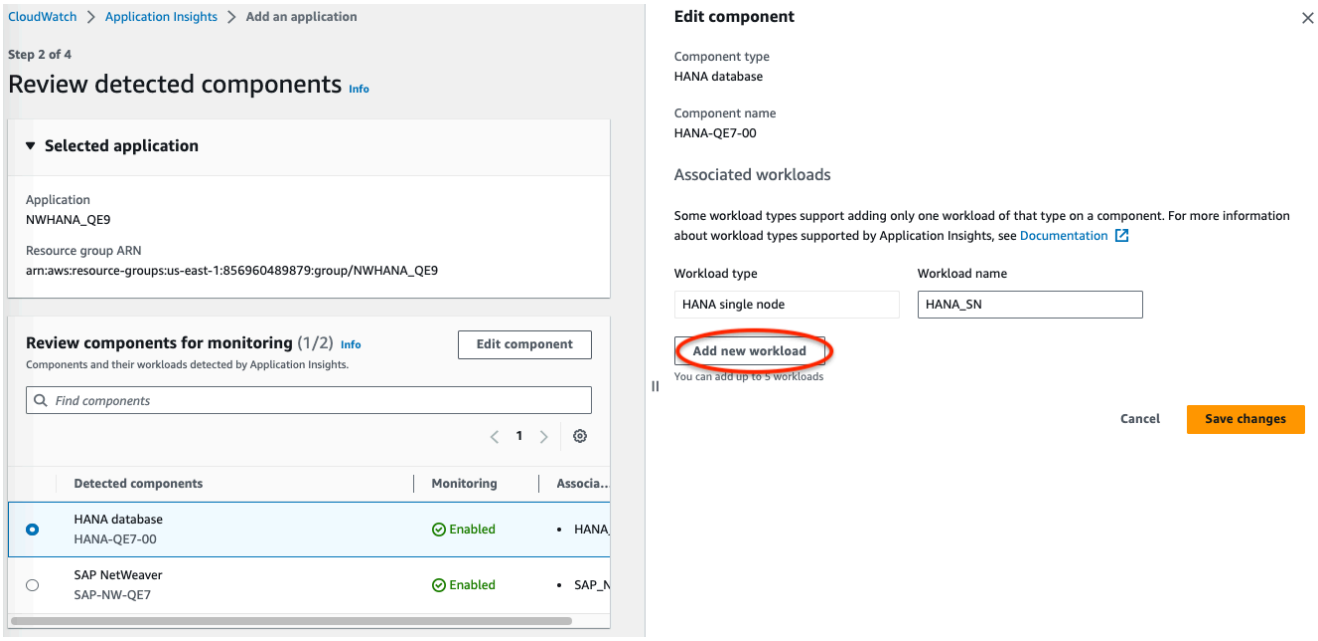
8. [Next] (次へ) を選択してモニターリングの設定を続行します。
9. [検出されたコンポーネントを確認] ページには、CloudWatch Application Insights によって自動的に検出されたモニターリング対象のコンポーネントとそれらのワークロードが一覧表示されます。

   a. 検出された SAP HANA 単一ノードワークロードを含むコンポーネントにワークロードを追加するには、コンポーネントを選択し [コンポーネントを編集] を選択します。

   > **Note**
   >
   > 検出された SAP HANA マルチノードまたは HANA 高可用性ワークロードを含むコンポーネントは、1 つのコンポーネントで 1 つのワークロードのみをサポートします。



   b. 新しいワークロードを追加するには、[新しいワークロードを追加] を選択します。



c. ワークロードの編集が完了したら、[変更の保存] を選択します。

10. [Next] を選択します。
11. [コンポーネントの詳細を指定] ページで、ユーザー名とパスワードを入力します。
12. アプリケーションのモニターリング設定を確認し [Submit] (送信) をクリックします。
13. アプリケーション詳細ページが開き、[アプリケーションの概要] および [モニタリング対象コンポーネントとワークロード] と [モニタリング対象ではないコンポーネントとワークロード] のリストを表示できます。コンポーネントまたはワークロードの横にあるラジオボタンをオンにすると、[設定履歴]、[ログパターン] 、および作成したタグも表示できます。設定を送信すると、アカウントで SAP HANA システムのすべてのメトリクスとアラームがデプロイされます。これには最大 2 時間かかります。

SAP HANA データベースのモニターリングの管理

次の手順を実行して、SAP HANA データベースのユーザー認証情報、メトリクス、およびログパスを管理できます。

1. CloudWatch コンソールを開きます。
2. 左側のナビゲーションペインで、[Insights] の下にある [Application Insights] を選択します。
3. [Application Insights] ページには、Application Insights でモニターリングするアプリケーションのリストと、各アプリケーションのモニターリングのステータスが表示されます。

4. [Monitored components] (モニターリングされているコンポーネント) で、コンポーネント名の横にあるラジオボタンをオンにします。次に、[Manage monitoring] (モニターリングの管理) を選択します。
5. [EC2 instance group logs] (EC2 インスタンスグループのログ) では、既存のログパス、ログパターンのセット、ロググループ名を更新できます。さらに、追加で Application logs (アプリケーションログ) を最大 3 つまで追加できます。
6. [Metrics] (メトリクス) では、要件に応じて SAP HANA メトリクスを選択できます。SAP HANA のメトリクス名は、hanadb で始まります。コンポーネントごとに最大 40 個のメトリクスを追加できます。
7. [HANA configuration] (HANA の設定) で、SAP HANA データベースのパスワードとユーザー名を入力します。これらは、Amazon CloudWatch エージェントが SAP HANA データベースに接続するために使用されるユーザー名とパスワードです。
8. [Custom alarms] (カスタムアラーム) では、CloudWatch Application Insights によってモニターリングするアラームを追加できます。
9. アプリケーションのモニターリング設定を確認し、[Submit] (送信) をクリックします。設定を送信すると、アカウントで SAP HANA システムのすべてのメトリクスとアラームが更新されます。これには最大 2 時間かかります。

CloudWatch Application Insights で検出された SAP HANA における問題の表示とトラブルシューティング

次のセクションでは、Application Insights で SAP HANA のモニターリングを設定する際に発生する一般的なトラブルシューティングのシナリオを解決する手順について説明します。

トラブルシューティングのトピック

- SAP HANA データベースがメモリの割り当て制限に達しました
- ディスクに空き容量がないイベント
- SAP HANA バックアップの実行が停止しました

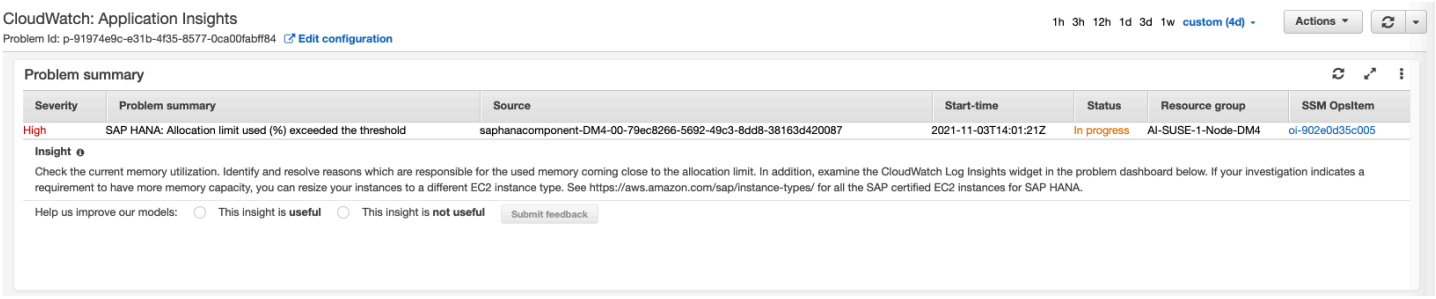
SAP HANA データベースがメモリの割り当て制限に達しました

説明

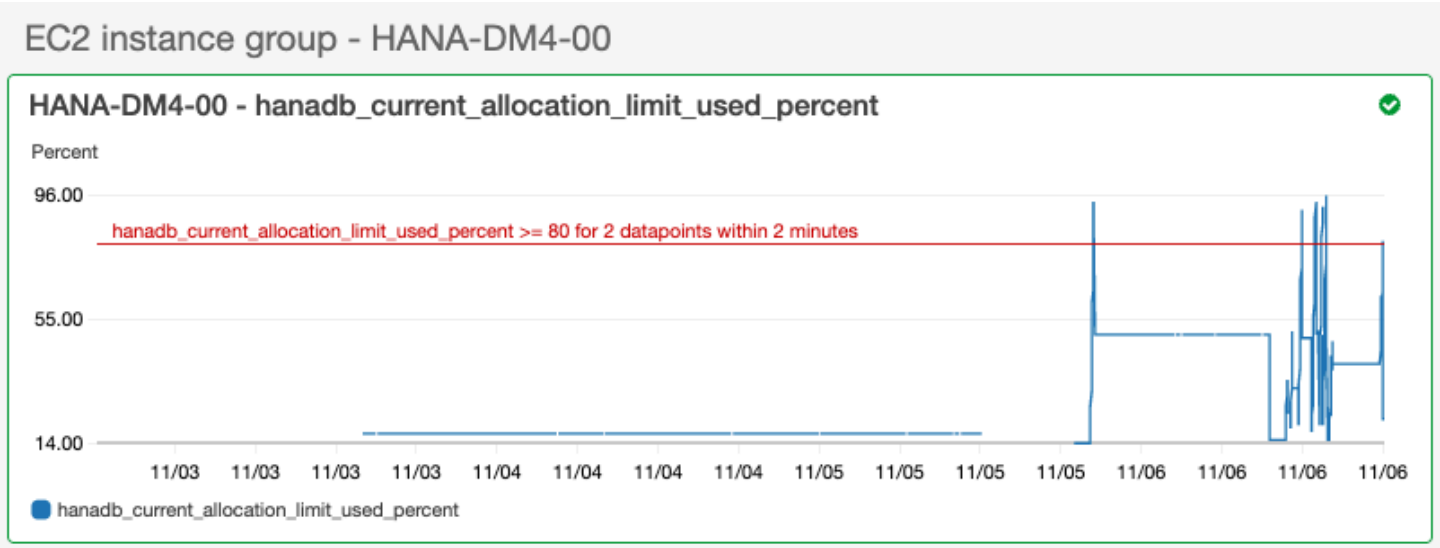
SAP HANA データベースによりバックアップされている SAP アプリケーションが高いメモリ負荷のために正常に動作せず、アプリケーションのパフォーマンスが低下しています。

解決方法

動的に作成されるダッシュボードを確認することで、問題を起こしているアプリケーション層を特定できます。ダッシュボードには、関連するメトリクスやログファイルのスニペットが表示されます。次の例では、SAP HANA システムにおける大きなデータロードが問題の原因である可能性があります。



使用されているメモリの割り当てが、合計のメモリの割り当て制限のしきい値の 80% を超えています。



ロググループに、メモリ不足のスキーム BNR-DATA とテーブル IMDBMASTER_30003 が表示されます。さらに問題の正確な時刻、現在のグローバルロケーションの制限、共有メモリ、コードサイズ、OOM の予約割り当てサイズが表示されます。

Log Group: SAP_HANA_TRACE-AI-SUSE-1-Node-DM4, Log Type: SAP_HANA_TRACE, AWS::SAPHANA.OutOfMemory

#	@timestamp	@message
1	2021-11-06T13:31:23.317Z	GLOBAL_ALLOCATION_LIMIT (GAL) = 55.78gb (59901001728b), SHARED_MEMORY = 567.77mb (595357696b), CODE_SIZE = 2.94gb (3162550272b), OOM_RESERVATION_ALLOCATOR_SIZE = 96.14mb (100810752b)
2	2021-11-06T13:31:23.316Z	[2867]{311260}[22/963854] 2021-11-06 13:00:44.999570 e OOM_Notification Statement.cc(04580) : oom exception occurred at 'imdbmaster:30003': conn_id=311260, stmt_id=1336853818011966, stmt_hash=17e1ccc2b5f460604ceae8c98690fd01, sql=CAL…
3	2021-11-06T13:31:23.316Z	[3033]{311513}[22/967162] 2021-11-06 13:31:17.163640 e Memory mmReportMemoryProblems.cpp(01805) : OUT OF MEMORY occurred.
4	2021-11-06T13:31:23.316Z	Current callstack: 1: 0x00007f824538dd35 in MemoryManager::PoolAllocator::notifyOOMImpl(unsigned long, unsigned long, bool, ltt::allocation_failure_type, bool)+0x1b1 at mmPoolAllocator.cpp:2284 (libhdbbasis.so) 2: 0x00007f824524a7ad …
5	2021-11-06T13:31:23.316Z	[2822]{-1}[-1/-1] 2021-11-06 13:31:17.175597 e Memory mmReportMemoryProblems.cpp(01805) : OUT OF MEMORY occurred.
6	2021-11-06T13:31:23.316Z	Current callstack: 1: 0x00007f824538dd35 in MemoryManager::PoolAllocator::notifyOOMImpl(unsigned long, unsigned long, bool, ltt::allocation_failure_type, bool)+0x1b1 at mmPoolAllocator.cpp:2284 (libhdbbasis.so) 2: 0x00007f824524a7ad …
7	2021-11-06T13:31:23.316Z	GLOBAL_ALLOCATION_LIMIT (GAL) = 55.78gb (59901001728b), SHARED_MEMORY = 567.77mb (595357696b), CODE_SIZE = 2.94gb (3162550272b), OOM_RESERVATION_ALLOCATOR_SIZE = 96.14mb (100810752b)
8	2021-11-06T13:31:17.318Z	GLOBAL_ALLOCATION_LIMIT (GAL) = 55.78gb (59901001728b), SHARED_MEMORY = 567.77mb (595357696b), CODE_SIZE = 2.94gb (3162550272b), OOM_RESERVATION_ALLOCATOR_SIZE = 96.14mb (100810752b)
9	2021-11-06T13:31:17.317Z	GLOBAL_ALLOCATION_LIMIT (GAL) = 55.78gb (59901001728b), SHARED_MEMORY = 567.77mb (595357696b), CODE_SIZE = 2.94gb (3162550272b), OOM_RESERVATION_ALLOCATOR_SIZE = 96.14mb (100810752b)
10	2021-11-06T13:31:17.317Z	[3033]{311513}[22/967162] 2021-11-06 13:31:17.100223 w Memory mmPoolAllocator.cpp(01212) : Out of memory for Pool/PersistenceManager/PersistentSpace/DefaultLPA/DataPage, size 16777216B, alignment=4096B, flags 0x0, reason GLOBAL_ALLOC…
11	2021-11-06T13:31:17.317Z	GLOBAL_ALLOCATION_LIMIT (GAL) = 55.78gb (59901001728b), SHARED_MEMORY = 567.77mb (595357696b), CODE_SIZE = 2.94gb (3162550272b), OOM_RESERVATION_ALLOCATOR_SIZE = 96.14mb (100810752b)
12	2021-11-06T13:31:17.317Z	[3033]{311513}[22/967162] 2021-11-06 13:31:17.163640 e Memory mmReportMemoryProblems.cpp(01805) : OUT OF MEMORY occurred.
13	2021-11-06T13:31:17.317Z	Current callstack: 1: 0x00007f824538dd35 in MemoryManager::PoolAllocator::notifyOOMImpl(unsigned long, unsigned long, bool, ltt::allocation_failure_type, bool)+0x1b1 at mmPoolAllocator.cpp:2284 (libhdbbasis.so) 2: 0x00007f824524a7ad …
14	2021-11-06T13:31:17.317Z	[2822]{-1}[-1/-1] 2021-11-06 13:31:17.170707 w Memory mmPoolAllocator.cpp(01212) : Out of memory for Pool/malloc/libhdbbasement.so, size 422808, alignment=8B, flags 0x0, reason GLOBAL_ALLOCATION_LIMIT
15	2021-11-06T13:31:17.317Z	[2822]{-1}[-1/-1] 2021-11-06 13:31:17.175597 e Memory mmReportMemoryProblems.cpp(01805) : OUT OF MEMORY occurred.
16	2021-11-06T13:31:17.317Z	Current callstack: 1: 0x00007f824538dd35 in MemoryManager::PoolAllocator::notifyOOMImpl(unsigned long, unsigned long, bool, ltt::allocation_failure_type, bool)+0x1b1 at mmPoolAllocator.cpp:2284 (libhdbbasis.so) 2: 0x00007f824524a7ad …
17	2021-11-06T13:31:17.317Z	GLOBAL_ALLOCATION_LIMIT (GAL) = 55.78gb (59901001728b), SHARED_MEMORY = 567.77mb (595357696b), CODE_SIZE = 2.94gb (3162550272b), OOM_RESERVATION_ALLOCATOR_SIZE = 96.14mb (100810752b)
18	2021-11-06T13:31:16.317Z	GLOBAL_ALLOCATION_LIMIT (GAL) = 55.78gb (59901001728b), SHARED_MEMORY = 567.77mb (595357696b), CODE_SIZE = 2.94gb (3162550272b), OOM_RESERVATION_ALLOCATOR_SIZE = 96.14mb (100810752b)
19	2021-11-06T13:31:16.317Z	GLOBAL_ALLOCATION_LIMIT (GAL) = 55.78gb (59901001728b), SHARED_MEMORY = 567.77mb (595357696b), CODE_SIZE = 2.94gb (3162550272b), OOM_RESERVATION_ALLOCATOR_SIZE = 96.14mb (100810752b)

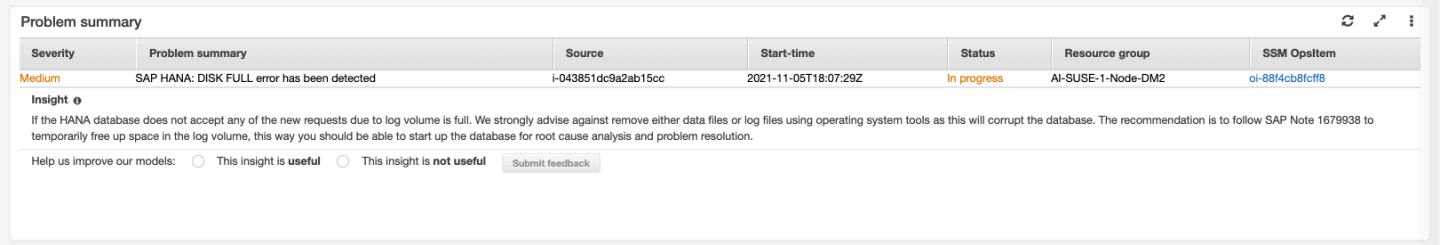
ディスクに空き容量がないイベント

説明

SAP HANA データベースによってバックアップされている SAP アプリケーションが応答を停止し、データベースにアクセスできません。

解決方法

動的に作成されるダッシュボードを確認することで、問題が起きているデータベース層を特定できます。ダッシュボードには、関連するメトリクスやログファイルのスニペットが表示されます。次の例では、管理者が自動ログバックアップを有効にできなかったため、sap/hana/log ディレクトリがいっぱいになったことが問題である可能性があります。



問題のあるダッシュボードのロググループのウィジェットには、DISKFULL イベントが表示されます。

Log Group: SAP_HANA_TRACE-AI-SUSE-1-Node-DM2, Log Type: SAP_HANA_TRACE, AWS::SAPHANA.DiskFull

#	@timestamp	@message
1	2021-11-06T18:00:20.072Z	[26768]{-1}[-1/-1] 2021-11-06 18:00:16.556583 i EventHandler LocalFileCallback.cpp(00517) : [DISKFULL] restarting queue with 1 requests
	@ingestionTime	1636221622489
	@log	[redacted]:SAP_HANA_TRACE-AI-SUSE-1-Node-DM2
	@logStream	i-[redacted]
	@message	[26768]{-1}[-1/-1] 2021-11-06 18:00:16.556583 i EventHandler LocalFileCallback.cpp(00517) : [DISKFULL] restarting queue with 1 requests
	@timestamp	1636221620072

SAP HANA バックアップの実行が停止しました

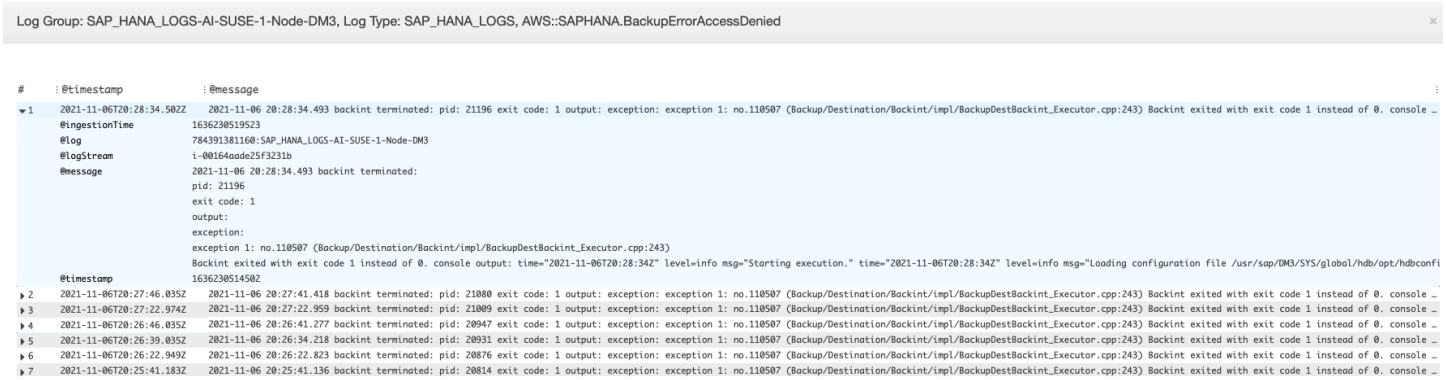
説明

SAP HANA データベースによってバックアップされている SAP アプリケーションが停止しました。

解決方法

動的に作成されるダッシュボードを確認することで、問題が起きているデータベース層を特定できます。ダッシュボードには、関連するメトリクスやログファイルのスニペットが表示されます。

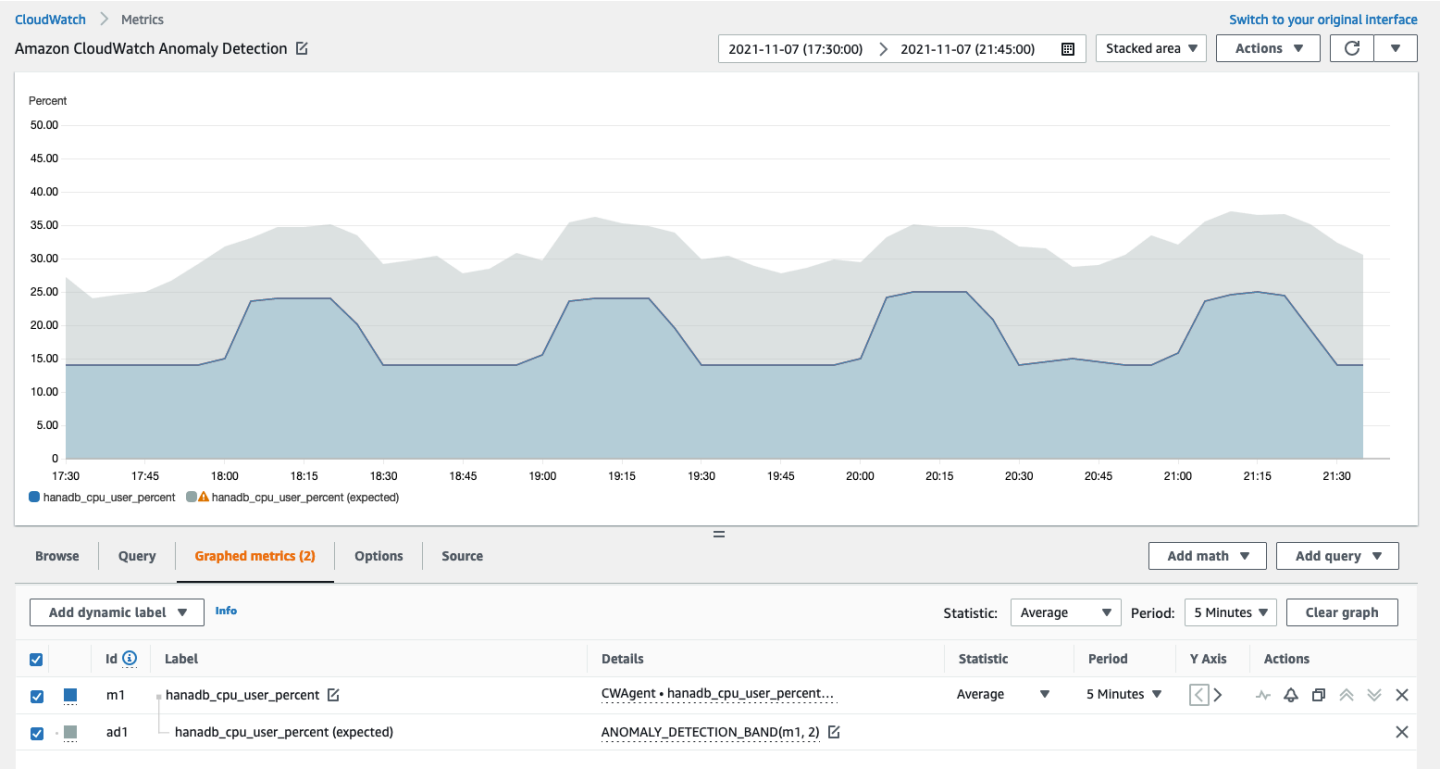
問題のあるダッシュボードのロググループのウィジェットには、ACCESS DENIED イベントが表示されます。これには、S3 バケット、S3 バケットフォルダー、S3 バケットリージョンなどの追加情報が含まれます。



SAP HANA の異常検出

スレッド数など特定の SAP HANA メトリクスについては、CloudWatch により統計および機械学習アルゴリズムが適用され、しきい値が定義されます。これらのアルゴリズムにより、SAP HANA データベースのメトリクスが継続的に分析され、正常なベースラインが決定します。このため、ユーザーの介入が最小限でも異常を検出できます。アルゴリズムにより、異常検出モデルが生成されます。このモデルにより、メトリクスの正常な動作を表す想定値の範囲が生成されます。

異常検出アルゴリズムは、メトリクスの季節的な変化と傾向の変化を考慮します。季節的な変化は、次の例に示す SAP HANA の CPU 使用率のように、時間単位、日単位、週単位のいずれかになります。



モデルを作成した後、CloudWatch の異常検出ではモデルの継続的な評価および調整が行われ、可能な限り正確であることが確認されます。これには、メトリクス値が時間の経過とともに変化するか、または突然変化するかを調整するためのモデルの再トレーニングが含まれます。また、季節的、スパイク、スパースなメトリクスのモデルを改善するための予測変数も含まれます。

SAP HANA 向けの Application Insights のトラブルシューティング

このセクションでは、Application Insights ダッシュボードから返される一般的なエラーを解決するための手順について説明します。

60 を超える監視対象メトリクスを追加できない

出力では、次のエラーが表示されます。

```
Component cannot have more than 60 monitored metrics
```

根本原因 – 現在の監視対象メトリクスの上限は、コンポーネントごとに 60 個です。

解決方法 – 制限を超えないように、必要のないメトリクスを削除します。

オンボーディングプロセス後に SAP メトリクスが表示されない

次の情報を参照して、オンボーディングプロセス後に SAP メトリクスがダッシュボードに表示されない理由を特定してください。最初のステップは、Amazon EC2 インスタンスの AWS Management Consoleまたはエクスポーターログを使用して、SAP メトリクスが表示されない理由をトラブルシューティングすることです。次に、エラー出力を確認して解決策を見つけます。

オンボーディング後に SAP メトリクスが表示されない理由をトラブルシューティングします。

Amazon EC2 インスタンスの AWS Management Consoleまたはエクスポーターログをトラブルシューティングに使用できます。

AWS Management Console

オンボーディング後に SAP メトリクスが表示されない問題をコンソールを使用してトラブルシューティングする

1. AWS Systems Manager コンソール (https://console.aws.amazon.com/systems-manager/) を開きます。
2. 左側のナビゲーションペインで、[ステートマネージャー] を選択します。
3. [関連付け] で、文書 `AWSEC2-ApplicationInsightsCloudwatchAgentInstallAndConfigure` のステータスを確認します。ステータスが `Failed` の場合は、[実行 ID] で失敗した ID を選択し、出力を表示します。
4. [関連付け] で、文書 `AWS-ConfigureAWSPackage` のステータスを確認します。ステータスが `Failed` の場合は、[実行 ID] で失敗した ID を選択し、出力を表示します。

Exporter logs from Amazon EC2 instance

オンボーディング後に SAP メトリクスが表示されない問題をエクスポーターログを使用してトラブルシューティングする

1. SAP HANA データベースが実行されている Amazon EC2 インスタンスに接続します。
2. 次のコマンドを使用して、`WORKLOAD_SHORT_NAME` に使用する正しい命名規則を見つけます。この省略名は次の 2 つのステップで使用します。

```
sudo systemctl | grep exporter
```

> **Note**
>
> Application Insights は、実行中のワークロードに応じて、サービス名にサフィックス WORKLOAD_SHORT_NAME を追加します。SAP HANA のシングルノード、マルチノード、高可用性デプロイの省略名は HANA_SN、HANA_MN、および HANA_HA です。

3. エクスポーターマネージャーのサービスログにエラーがないかチェックするには、WORKLOAD_SHORT_NAME を Step 2 で見つけた省略名に置き換えて、次のコマンドを実行します。

```
sudo journalctl -e --unit=prometheus-
hanadb_exporter_manager_WORKLOAD_SHORT_NAME.service
```

4. エクスポーターマネージャーのサービスログにエラーが表示されない場合は、次のコマンドを実行してエクスポーターサービスログにエラーがないか確認します。

```
sudo journalctl -e --unit=prometheus-hanadb_exporter_WORKLOAD_SHORT_NAME.service
```

オンボーディング後に SAP メトリクスが表示されない問題の一般的な根本原因を解決する

以下の例は、オンボーディング後に SAP メトリクスが表示されない問題の一般的な根本原因を解決する方法を示しています。

- 出力では、次のエラーが表示されます。

```
Reading json config file path: /opt/aws/amazon-cloudwatch-agent/etc/amazon-
cloudwatch-agent.d/default ...
Reading json config file path: /opt/aws/amazon-cloudwatch-agent/etc/
amazon-cloudwatch-agent.d/ssm_AmazonCloudWatch-ApplicationInsights-
SSMParameterForTESTCWEC2INSTANCEi0d88867f1f3e36285.tmp ...
2023/11/30 22:25:17 Failed to merge multiple json config files.
2023/11/30 22:25:17 Failed to merge multiple json config files.
2023/11/30 22:25:17 Under path : /metrics/append_dimensions | Error : Different
 values are specified for append_dimensions
2023/11/30 22:25:17 Under path : /metrics/metrics_collected/disk | Error : Different
 values are specified for disk
2023/11/30 22:25:17 Under path : /metrics/metrics_collected/mem | Error : Different
 values are specified for mem
```

```
2023/11/30 22:25:17 Configuration validation first phase failed. Agent version: 1.0.
 Verify the JSON input is only using features supported by this version.
```

解決策 — Application Insights は、既存の CloudWatch エージェントの設定ファイルの一部として事前設定されているものと同じメトリクスを設定しようとしています。/opt/aws/amazon-cloudwatch-agent/etc/amazon-cloudwatch-agent.d/ にある既存のファイルを削除するか、既存の CloudWatch エージェントの設定ファイルから競合の原因となっているメトリクスを削除します。

- 出力では、次のエラーが表示されます。

```
Unable to find a host with system database, for more info rerun using -v
```

解決策 — ユーザー名、パスワード、またはデータベースポートが間違っている可能性があります。ユーザー名、パスワード、およびポートが有効であることを確認し、その後オンボーディングプロセスに戻ります。

- 出力では、次のエラーが表示されます。

```
This hdbcli installer is not compatible with your Python interpreter
```

解決策 — Python 3.6 の次の例に示すように、pip3 と wheel をアップグレードします。

```
python3.6 -m pip install --upgrade pip setuptools wheel
```

- 出力では、次のエラーが表示されます。

```
Unable to install hdbcli using pip3. Please try to install it
```

解決策 — hdbclient の前提条件に従っていることを確認するか、pip3 に hdbclient を手動でインストールします。

- 出力では、次のエラーが表示されます。

```
Package 'boto3' requires a different Python: 3.6.15 not in '>= 3.7'
```

解決策 — このオペレーティングシステムのバージョンには Python 3.8 以降が必要です。Python 3.8 の前提条件を確認してインストールしてください。

- 出力では、次のいずれかのインストールエラーが表示されます。

```
Can not execute `setup.py` since setuptools is not available in the build environment
```

または

```
[SSL: CERTIFICATE_VERIFY_FAILED]
```

解決策 – 次の例に示すように、SUSE Linux コマンドを使用して Python をインストールします。次の例では、最新版の Python 3.8 をインストールします。

```
wget https://www.python.org/ftp/python/3.8.<LATEST_RELEASE>/
Python-3.8.<LATEST_RELEASE>.tgz
tar xf Python-3.*
cd Python-3.*/
sudo zypper install make gcc-c++ gcc automake autoconf libtool
sudo zypper install zlib-devel
sudo zypper install libopenssl-devel libffi-devel
./configure --with-ensurepip=install
sudo make
sudo make install
sudo su
python3.8 -m pip install --upgrade pip setuptools wheel
```

チュートリアル: SAP NetWeaver のモニターリングを設定する

このチュートリアルでは、SAP NetWeaver のモニターリングを設定できるよう Amazon CloudWatch Application Insights を設定する方法を説明します。CloudWatch Application Insights の自動ダッシュボードを使用して、問題の詳細を可視化し、トラブルシューティングを加速させ、SAP NetWeaver アプリケーションサーバーでの解決までの平均時間 (MTTR) を短縮できます。

SAP Netweaver 用 CloudWatch Application Insights のトピック

- サポートされている環境
- サポートされるオペレーティングシステム
- 機能
- 前提条件
- SAP NetWeaver アプリケーションサーバーをモニターリングするように設定する

- SAP NetWeaver アプリケーションサーバーのモニターリングを管理する
- CloudWatch Application Insights で検出された SAP Netweaver に関する問題の表示とトラブルシューティング
- SAP Netweaver 用 Application Insights のトラブルシューティング

サポートされている環境

CloudWatch Application Insights では、次のシステムおよびパターン用に AWS リソースのデプロイがサポートされています。

- SAP NetWeaver 標準システムデプロイ
- 複数の Amazon EC2 インスタンスでの SAP NetWeaver Distributed デプロイ。
- クロス AZ SAP Netweaver の高可用性セットアップ – SUSE/RHEL クラスタリングにより 2 つのアベイラビリティーゾーンでの高可用性が設定されている SAP Netweaver。

サポートされるオペレーティングシステム

SAP Netweaver 用 CloudWatch Application Insights は、次のオペレーティングシステム上でサポートされています。

- Oracle Linux 8
- Red Hat Enterprise Linux 7.6
- Red Hat Enterprise Linux 7.7
- Red Hat Enterprise Linux 7.9
- Red Hat Enterprise Linux 8.1
- Red Hat Enterprise Linux 8.2
- Red Hat Enterprise Linux 8.4
- Red Hat Enterprise Linux 8.6
- SAP 用 SUSE Linux Enterprise Server 15
- SAP 用 SUSE Linux Enterprise Server 15 SP1
- SAP 用 SUSE Linux Enterprise Server 15 SP2
- SAP 用 SUSE Linux Enterprise Server 15 SP3
- SAP 用 SUSE Linux Enterprise Server 15 SP4
- SAP 用 SUSE Linux Enterprise Server 12 SP4

- SAP 用 SUSE Linux Enterprise Server 12 SP5
- High Availability パターンを除く SUSE Linux Enterprise Server 15
- High Availability パターンを除く SUSE Linux Enterprise Server 15 SP1
- High Availability パターンを除く SUSE Linux Enterprise Server 15 SP2
- High Availability パターンを除く SUSE Linux Enterprise Server 15 SP3
- High Availability パターンを除く SUSE Linux Enterprise Server 15 SP4
- High Availability パターンを除く SUSE Linux Enterprise Server 12 SP4
- High Availability パターンを除く SUSE Linux Enterprise Server 12 SP5

機能

SAP NetWeaver 7.0x—7.5x (ABAP Platform を含む) 用の CloudWatch Application Insights には、次の機能が用意されています。

- SAP NetWeaver ワークロードの自動検出
- 静的しきい値に基づく SAP Netweaver アラームの自動作成
- SAP Netweaver のログパターンの自動認識
- SAP NetWeaver 用のヘルスダッシュボード
- SAP NetWeaver 用の問題ダッシュボード

前提条件

CloudWatch Application Insights で SAP Netweaver を設定するには、以下を前もって実行する必要があります。

- AWS Systems Manager の有効化 – Amazon EC2 インスタンスに SSM Agent をインストールし、インスタンスを SSM に対して有効にします。SSM Agent のインストールに関する詳細については、「AWS Systems Manager ユーザーガイド」の「AWS Systems Manager のセットアップ」を参照してください。
- Amazon EC2 インスタンスロール – SAP NetWeaver モニターリングを設定するには、次の Amazon EC2 インスタンスロールをアタッチする必要があります。
  - Systems Manager を有効にする `AmazonSSMManagedInstanceCore` ロールをアタッチする必要があります。詳細については、「AWS Systems Manager アイデンティティベースのポリシーの例」を参照してください。

  - インスタンスメトリクスとログが CloudWatch を介して出力されるように `CloudWatchAgentServerPolicy` ポリシーをアタッチする必要があります。詳細については、「CloudWatch エージェントで使用する IAM ロールとユーザーを作成する」を参照してください

- AWS リソースグループ – アプリケーションを CloudWatch Application Insights にオンボードするには、アプリケーションスタックで使用されるすべての関連する AWS リソースを含むリソースグループを作成する必要があります。これには、SAP NetWeaver アプリケーションサーバーを実行する Amazon EC2 インスタンス、Amazon EFS、および Amazon EBS ボリュームが含まれます。1 つのアカウントに複数の SAP NetWeaver システムがある場合は、各 SAP Netweaver システムの AWS リソースを含む 1 つのリソースグループを作成することをお勧めします。リソースグループの作成についての詳細は、AWS Resource Groups とタグのユーザーガイドを参照してください。
- IAM アクセス許可 – 管理アクセスが許可されていないユーザーについては、Application Insights でサービスにリンクされたロールを作成できる AWS Identity and Access Management (IAM) ポリシーを作成し、ユーザーの ID にアタッチする必要があります。IAM ポリシーを作成する方法については、「IAM ポリシー」を参照してください。
- サービスにリンクされたロール – Application Insights は AWS Identity and Access Management (IAM) サービスにリンクされたロールを使用します。Application Insights コンソールで新しい Application Insights アプリケーションを作成する際、サービスにリンクされたロールが作成されます。詳細については、「CloudWatch Application Insights のサービスにリンクされたロールの使用」を参照してください。
- Amazon CloudWatch エージェント – Application Insights は、CloudWatch エージェントをインストールして設定します。CloudWatch エージェントがインストールされている場合、Application Insights は設定をそのまま維持します。マージ競合を避けるため、既存の CloudWatch エージェント設定ファイルから、Application Insights で使用するリソースの設定を削除してください。詳細については、「 CloudWatch エージェント設定ファイルを手動で作成または編集する」を参照してください。

SAP NetWeaver アプリケーションサーバーをモニターリングするように設定する

SAP NetWeaver アプリケーションサーバーのモニターリングを設定するには、次の手順に従います。

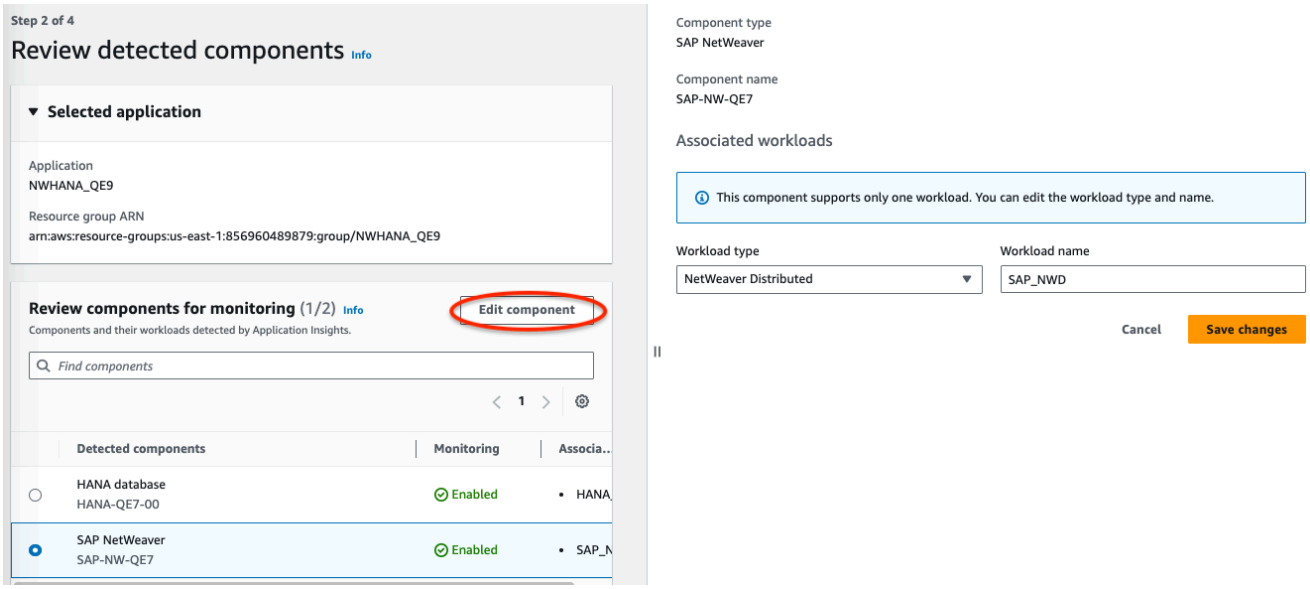
モニターリングを設定するには

1. CloudWatch コンソールを開きます。

2. 左側のナビゲーションペインで、[Insights] (インサイト) の下にある [Application Insights] を選択します。
3. [Application Insights] ページには、Application Insights でモニターリングするアプリケーションのリストと、各アプリケーションのモニターリングのステータスが表示されます。右上隅の [Add an application] (アプリケーションの追加) を選択します。
4. [Specify application details] (アプリケーションの詳細の指定) ページで、[Resource group] (リソースグループ) のドロップダウンリストから、SAP Netweaver リソースを含む作成済みの AWS リソースグループを選択します。アプリケーションのリソースグループを作成していない場合は、[Resource group] (リソースグループ) のドロップダウンリストから、[Create new resource group] (新しいリソースグループの作成) を選択して作成できます。
5. [Automatic monitoring of new resources] (新しいリソースの自動モニターリング) でチェックボックスをオンにすると、オンボーディング後にアプリケーションのリソースグループに追加されたリソースが Application Insights で自動的にモニターリングされるようになります。
6. [Monitor EventBridge events] (EventBridge イベントのモニターリング) で、CloudWatch Events と Application Insights のモニターリングを統合できるチェックボックスをオンにすると、Amazon EBS、Amazon EC2、AWS CodeDeploy、Amazon ECS、AWS Health APIs and Notifications、Amazon RDS、Amazon S3、および AWS Step Functions からのインサイトを取得できます。
7. 選択したアプリケーションの問題が検出されたときにそれらを表示し、通知を受け取るには、[AWS Systems Manager OpsCenter と統合する] にある [修正アクションのため AWS Systems Manager OpsCenter OpsItems を生成する] 横のチェックボックスをオンにします。AWS リソースに関連する OpsItems というオペレーション作業項目を解決するために実行されたオペレーションを追跡するには、SNS トピックの ARN を指定します。
8. オプションでタグを入力して、リソースを特定し整理できます。CloudWatch Application Insights では、タグベースのリソースグループと AWS CloudFormation スタックベースのリソースグループ (Application Auto Scaling グループを除く) の両方がサポートされています。詳細については、AWS Resource Groups とタグユーザーガイドの「Tag Editor」を参照してください。
9. 検出されたコンポーネントを確認するには、[次へ] を選択します。
10. [検出されたコンポーネントを確認] ページには、CloudWatch Application Insights によって自動的に検出されたモニターリング対象のコンポーネントとそれらのワークロードが一覧表示されます。

    - ワークロードの種類と名前を編集するには、[コンポーネントを編集] を選択します。

> **Note**
>
> 検出された NetWeaver Distributed ワークロードまたは NetWeaver High Availability ワークロードを含むコンポーネントは、1 つのコンポーネントで 1 つのワークロードのみをサポートします。



11. [Next] を選択します。
12. [Specify component details] (コンポーネントの詳細の指定) ページで [Next] (次へ) を選択します。
13. アプリケーションのモニターリング設定を確認し [送信] を選択します。
14. アプリケーションの詳細ページが開き、[アプリケーションの概要]、[ダッシュボード]、[コンポーネント] 、および [ワークロード] を表示できます。また、コンポーネントの横にあるラジオボタンをオンにすると、[Configuration history] (設定履歴)、[Log patterns] (ログのパターン)、および作成した [Tags] (タグ) を確認できます。アプリケーションを送信すると、CloudWatch Application Insights は SAP NetWeaver システムのすべてのメトリクスとアラームをデプロイします。デプロイには最大 1 時間かかります。

SAP NetWeaver アプリケーションサーバーのモニターリングを管理する

次の手順を使用して、SAP NetWeaver アプリケーションサーバーのモニターリングを管理します。

モニターリングを管理するには

1. CloudWatch コンソールを開きます。
2. 左側のナビゲーションペインで、[Insights] (インサイト) の下にある [Application Insights] を選択します。
3. [List view] (リストビュー) タブを選択します。
4. [Application Insights] ページには、Application Insights でモニターリングするアプリケーションのリストと、各アプリケーションのモニターリングのステータスが表示されます。
5. アプリケーションを選択します。
6. [Components] (コンポーネント) タブを選択します。
7. [Monitored components] (モニターリングされているコンポーネント) で、コンポーネント名の横にあるラジオボタンをオンにします。次に、[Manage monitoring] (モニターリングを管理) を選択します。
8. [Instance logs] (インスタンスログ) で、既存のログパス、ログパターンのセット、およびロググループ名を更新できます。さらに、追加で Application logs (アプリケーションログ) を最大 3 つまで追加できます。
9. [Metrics] (メトリクス) で、要件に応じて SAP Netweaver メトリクスを選択できます。SAP Netweaver のメトリクス名は、sap で始まります。コンポーネントごとに最大 40 個のメトリクスを追加できます。
10. [Custom alarms] (カスタムアラーム) では、CloudWatch Application Insights によってモニターリングするアラームを追加できます。
11. アプリケーションのモニターリング設定を確認し、[Save] (保存) を選択します。設定を送信すると、アカウントで SAP Netweaver システムのすべてのメトリクスとアラームが更新されます。

CloudWatch Application Insights で検出された SAP Netweaver に関する問題の表示とトラブルシューティング

次のセクションでは、Application Insights で SAP Netweaver のモニターリングを設定する際に発生する一般的なトラブルシューティングのシナリオを解決する手順について説明します。

トラブルシューティングのトピック

- SAP NetWeaver データベース接続の問題
- SAP NetWeaver アプリケーションの可用性に関する問題

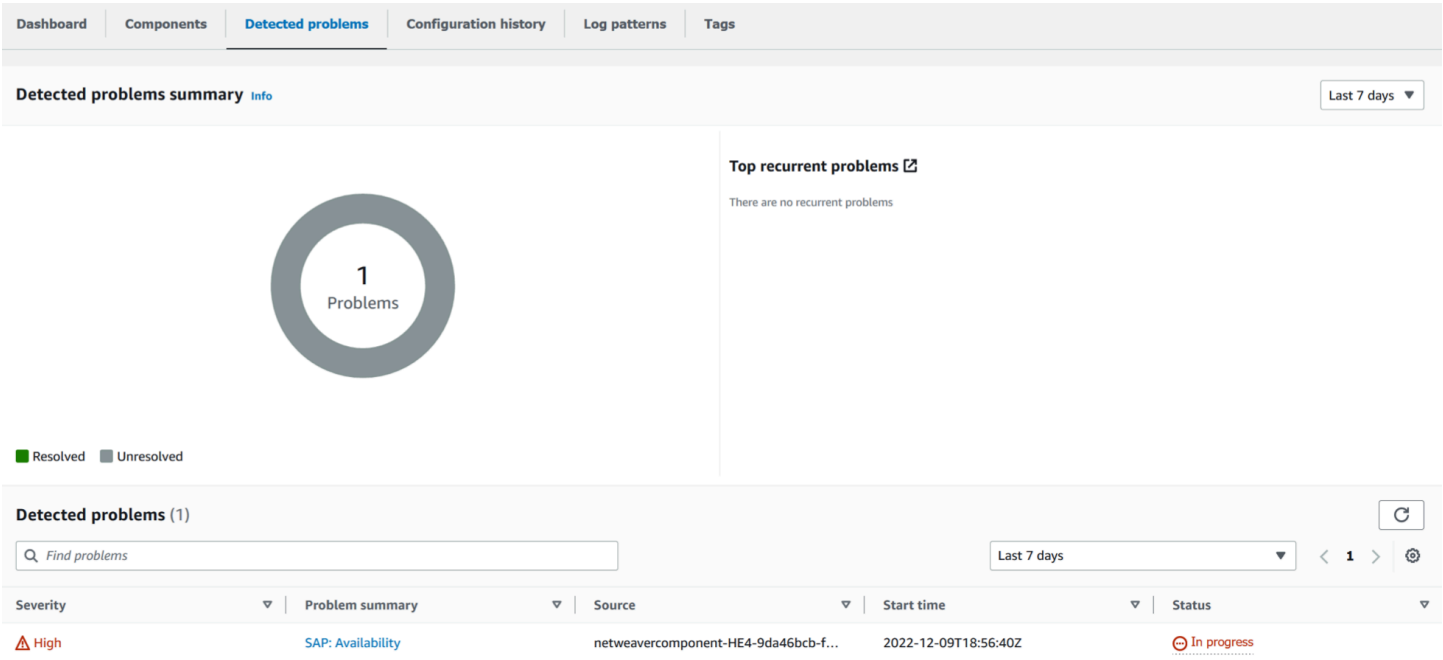
SAP NetWeaver データベース接続の問題

説明

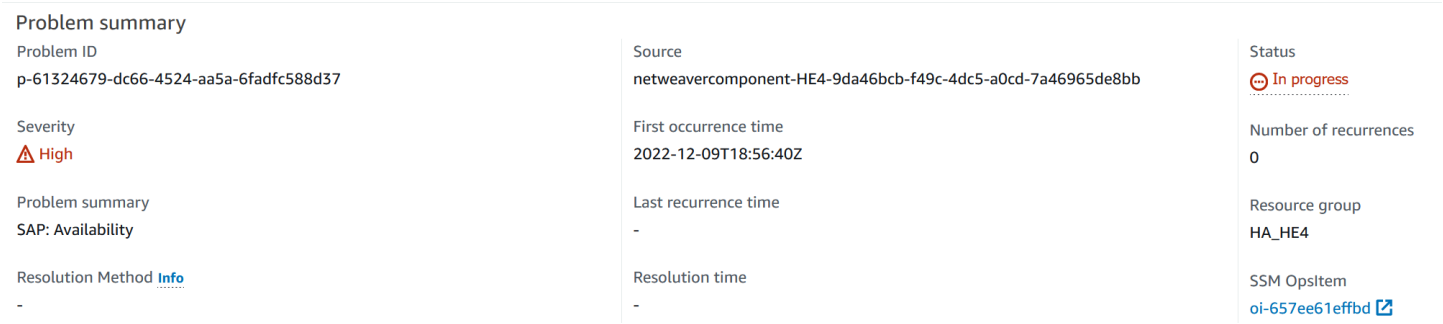
SAP NetWeaver アプリケーションにデータベース接続の問題が発生しています。

原因

接続の問題を特定するには、CloudWatch Application Insights コンソールに移動し、SAP NetWeaver Application Insights の問題ダッシュボードを確認します。[Problem summary] (問題の概要) でリンクを選択すると、特定の問題が表示されます。



次の例では、[Problem summary] (問題の概要) の下で「SAP: Availability」(SAP: 可用性) が問題として表示されています。

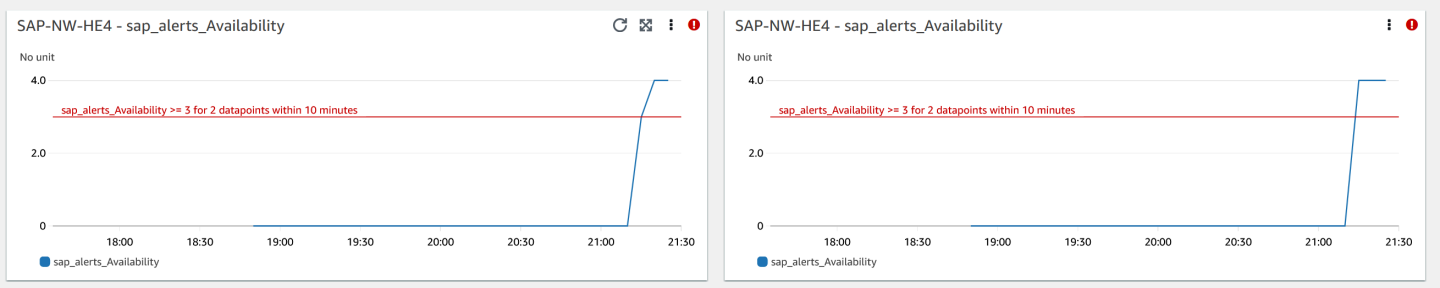


[Problem summary] (問題の概要) のすぐ後に、[Insight] (インサイト) セクションには、エラーに関する詳細なコンテキスト、および問題の原因に関する詳細情報を取得できる場所が表示されます。

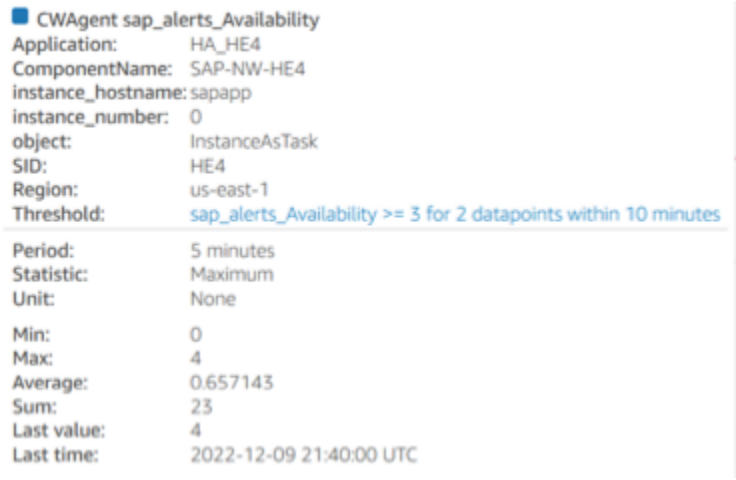
Insight Info

An availability issue with your SAP application server instance has been detected. Check SM21, SM50, SM51, SM66 and CCMS (RZ20) > InstanceAsTask > Availability.

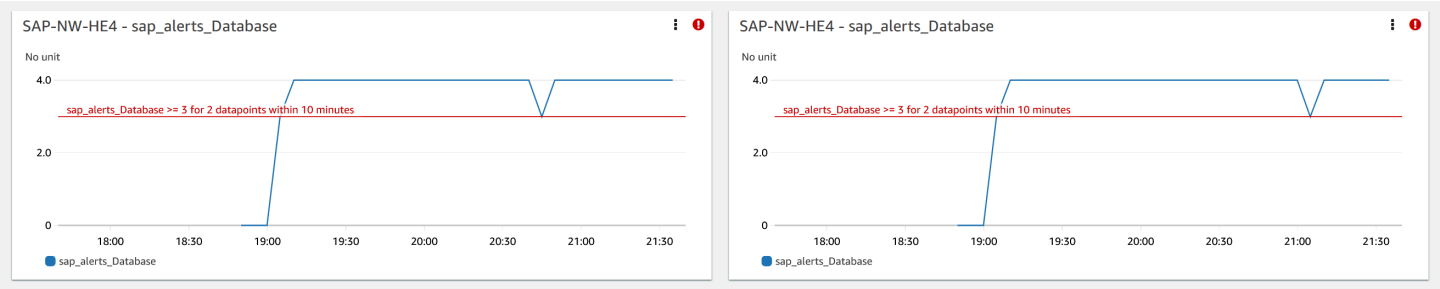
同じ問題ダッシュボードで、問題の検出によってグループ化された関連ログとメトリクスを表示すると、エラーの原因の切り分けに役立ちます。sap_alerts_Availability メトリクスは、SAP NetWeaver システムの可用性を経時的に追跡します。履歴追跡を使用して、メトリクスがエラー状態になったり、アラームのしきい値を超過したりした日時を関連付けることができます。次の例では、SAP NetWeaver システムに可用性の問題があります。この例では、2 つのアラームが表示されています。これは、SAP アプリケーションサーバーインスタンスが 2 つあり、インスタンスごとに 1 つのアラームが作成されているためです。



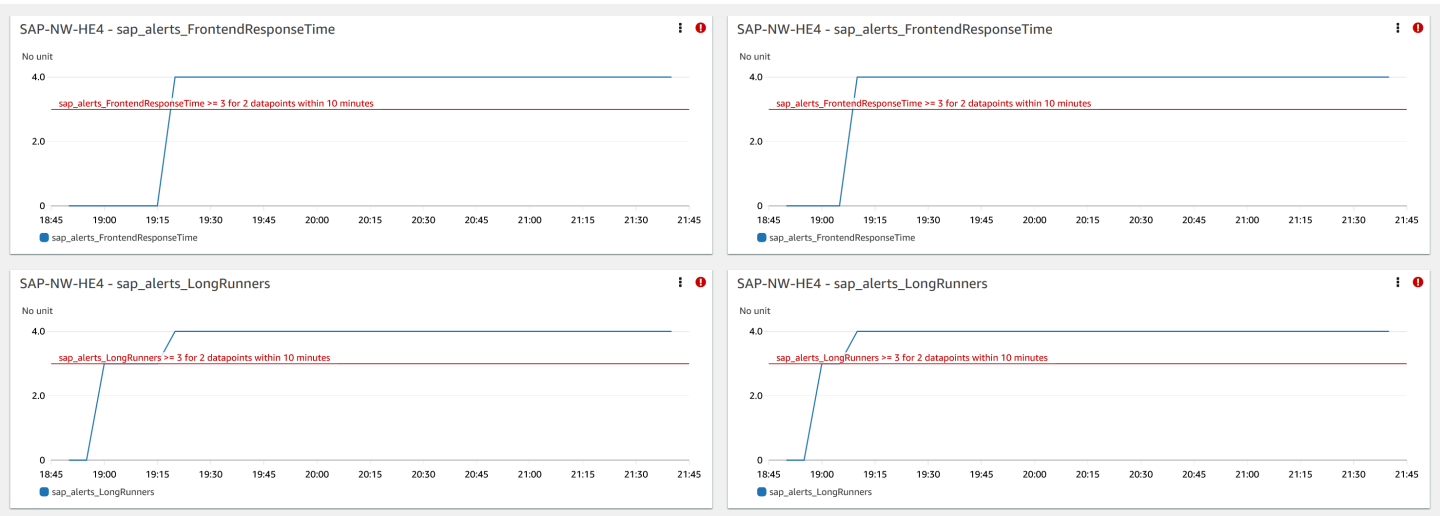
各アラームの詳細については、sap_alerts_Availability メトリクス名にカーソルを合わせてください。



次の例では、sap_alerts_Database メトリクスはデータベースレイヤーに問題や障害があることを示しています。このアラームは、SAP NetWeaver でデータベースへの接続やデータベースとの通信に問題が発生したことを示します。



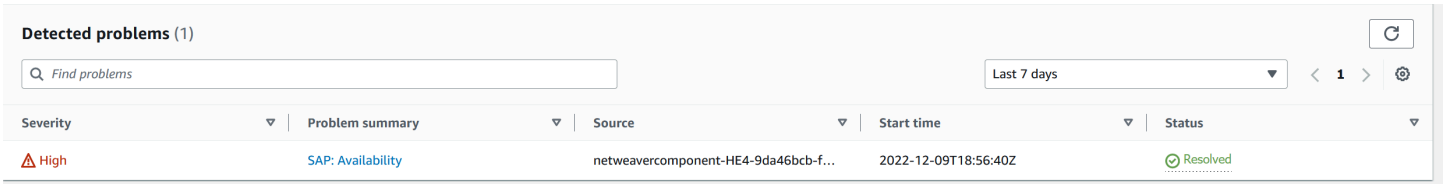
データベースは SAP NetWeaver の重要なリソースであるため、データベースに問題や障害が発生すると、関連するアラームが数多く表示されることがあります。次の例では、データベースが使用できないため、`sap_alerts_FrontendResponseTime` および `sap_alerts_LongRunners` メトリクスが開始されます。



解決方法

Application Insights は、検出された問題を 1 時間ごとにモニターリングします。SAP NetWeaver ログファイルに新しい関連ログエントリがない場合、古いログエントリは解決済みとして処理されます。CloudWatch アラームに関連するエラー条件をすべて修正する必要があります。エラー条件が修正された後、アラームとログが復旧するとアラームが解決されます。CloudWatch ログのエラーとアラームがすべて解決されると、Application Insights はエラーの検出を停止し、問題は 1 時間以内に自動的に解決されます。問題ダッシュボードに最新の問題が表示されるように、ログのエラー条件とアラームすべてを解決することをお勧めします。

次の例では、SAP Availability の問題が解決されています。



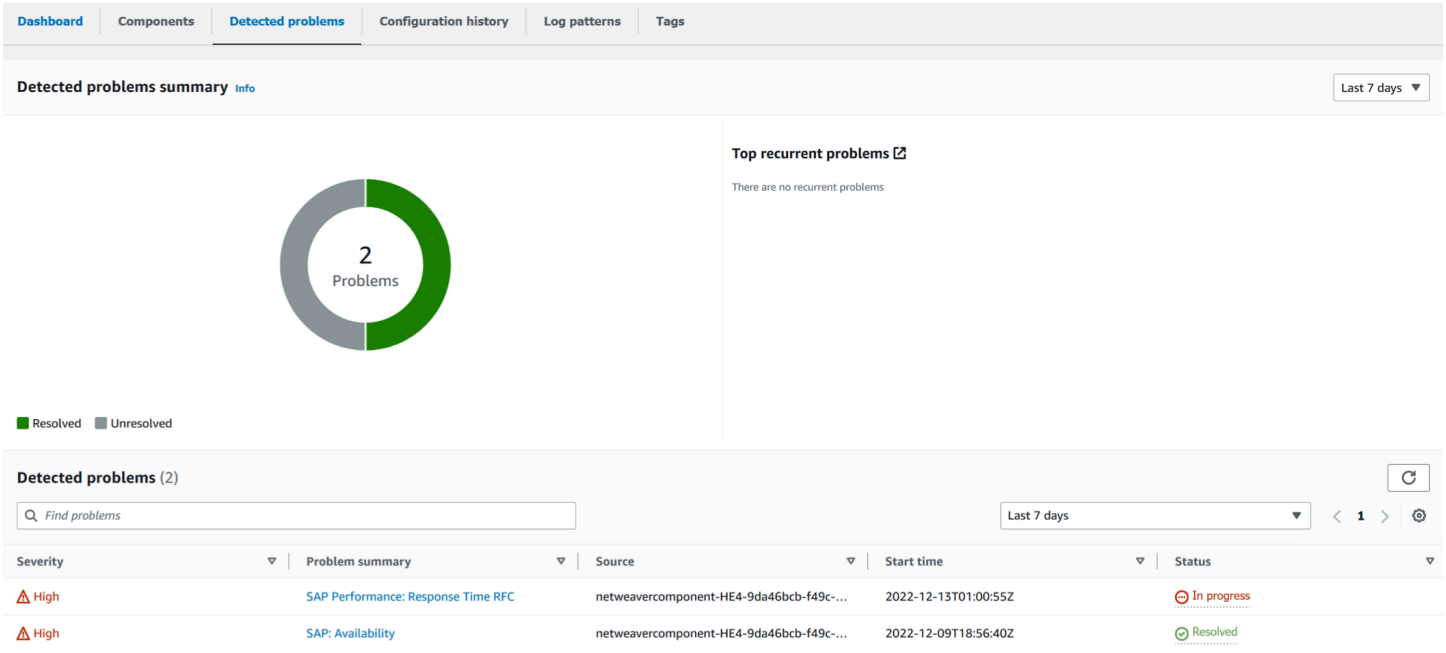
SAP NetWeaver アプリケーションの可用性に関する問題

説明

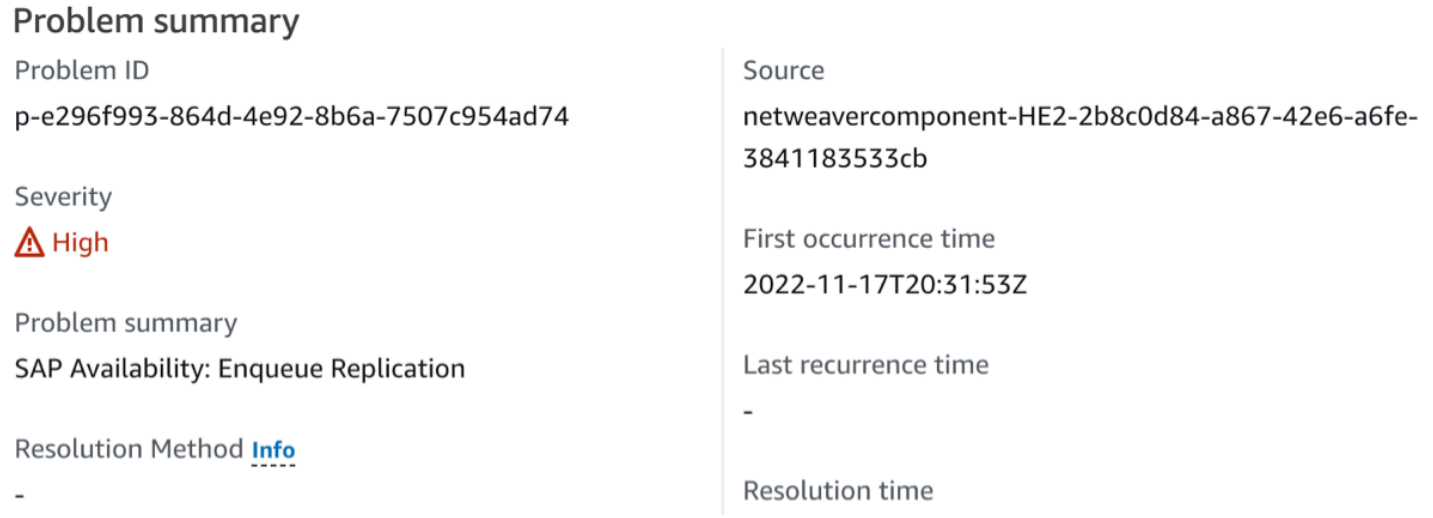
SAP NetWeaver High Availability Enqueue Replication が機能しなくなりました。

原因

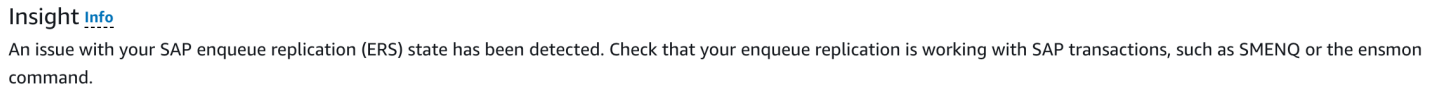
接続の問題を特定するには、CloudWatch Application Insights コンソールに移動し、SAP NetWeaver Application Insights の問題ダッシュボードを確認します。[Problem summary] (問題の概要) でリンクを選択すると、特定の問題が表示されます。



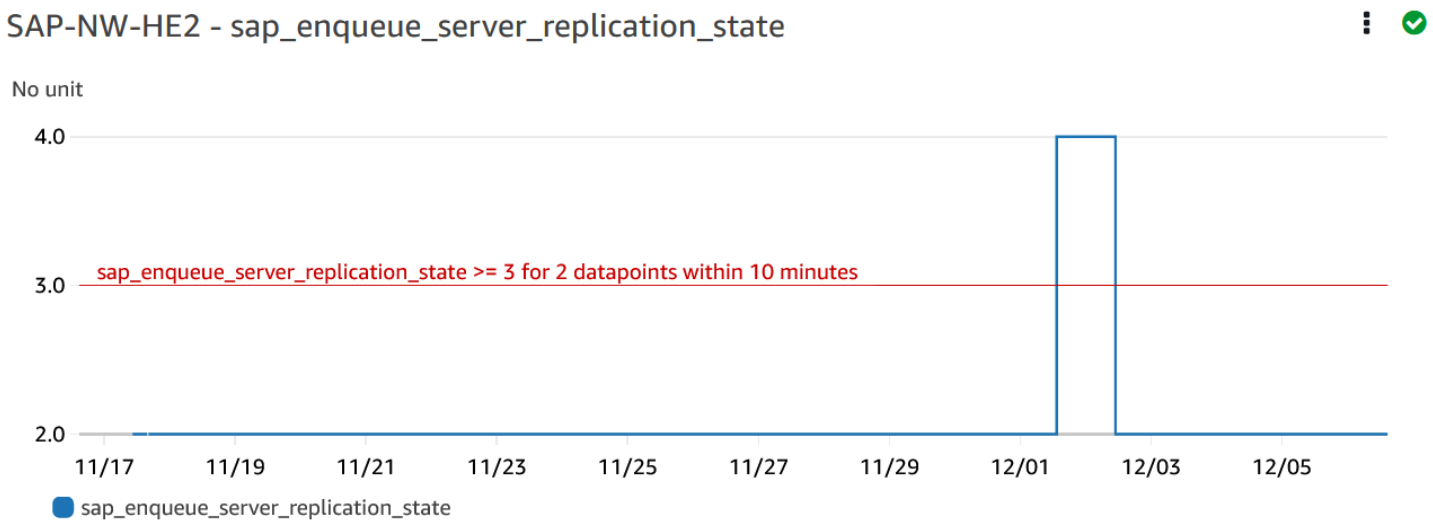
次の例では、[Problem summary] (問題の概要) の下で、「High Availability Enqueue Replication」が問題として表示されています。



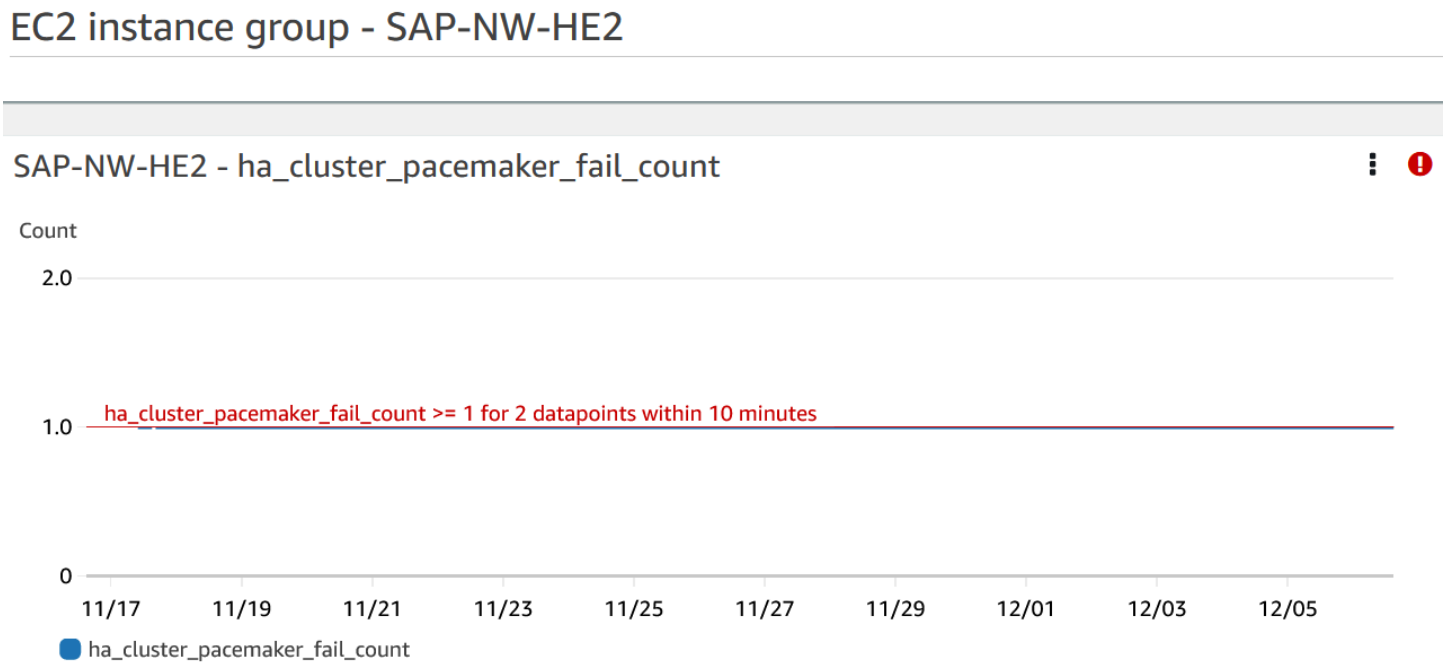
[Problem summary] (問題の概要) のすぐ後に、[Insight] (インサイト) セクションには、エラーに関する詳細なコンテキスト、および問題の原因に関する詳細情報を取得できる場所が表示されます。



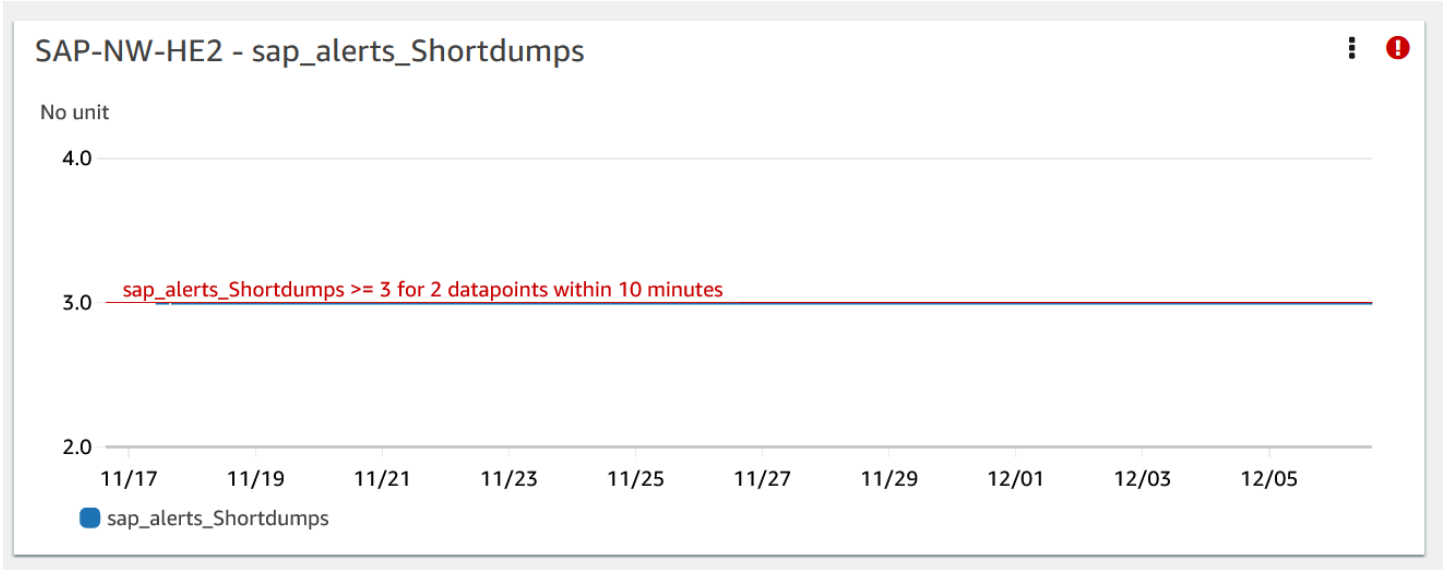
次の例は、エラーの原因の分離に役立つようにグループ化されたログとメトリクスが表示される問題ダッシュボードを示しています。sap_enqueue_server_replication_state メトリクスは値を経時的に追跡します。履歴追跡を使用して、メトリクスがエラー状態になったり、アラームのしきい値を超過したりした日時を関連付けることができます。



次の例では、ha_cluster_pacemaker_fail_count メトリクスは、高可用性ペースメーカークラスターでリソース障害が発生したことを示しています。フェイルカウントが 1 以上の特定のペースメーカーリソースは、コンポーネントダッシュボードで特定されます。

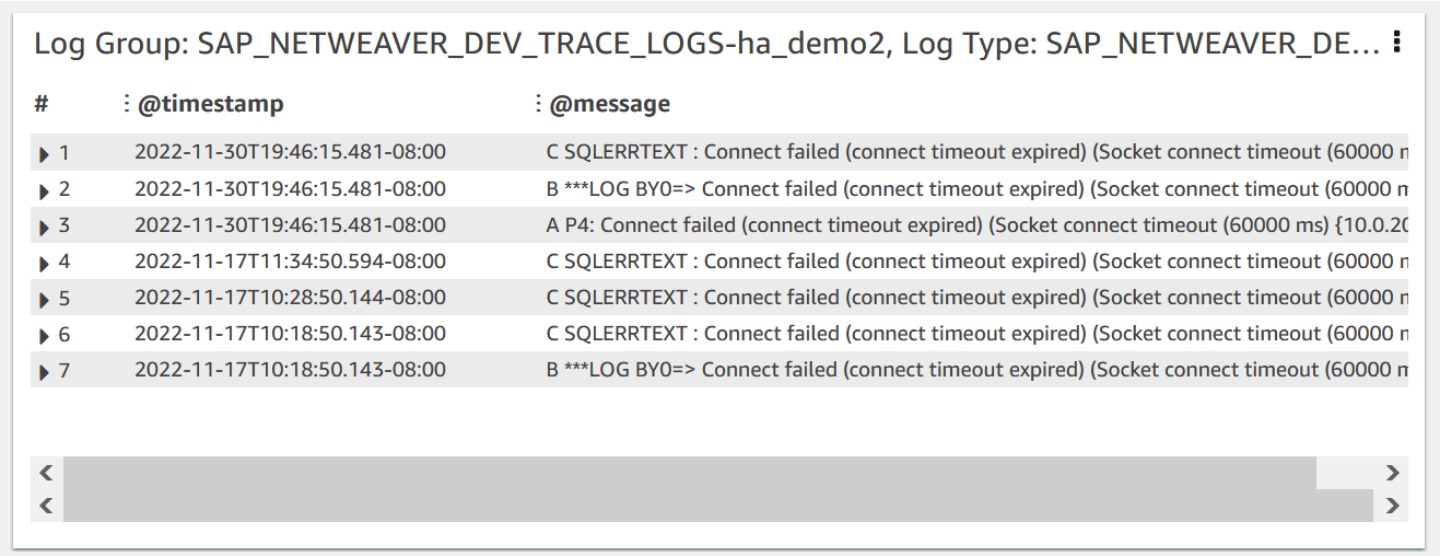


次の例は、問題が検出されたときに SAP アプリケーションのパフォーマンスが低下したことを示す `sap_alerts_Shortdumps` メトリクスを示しています。

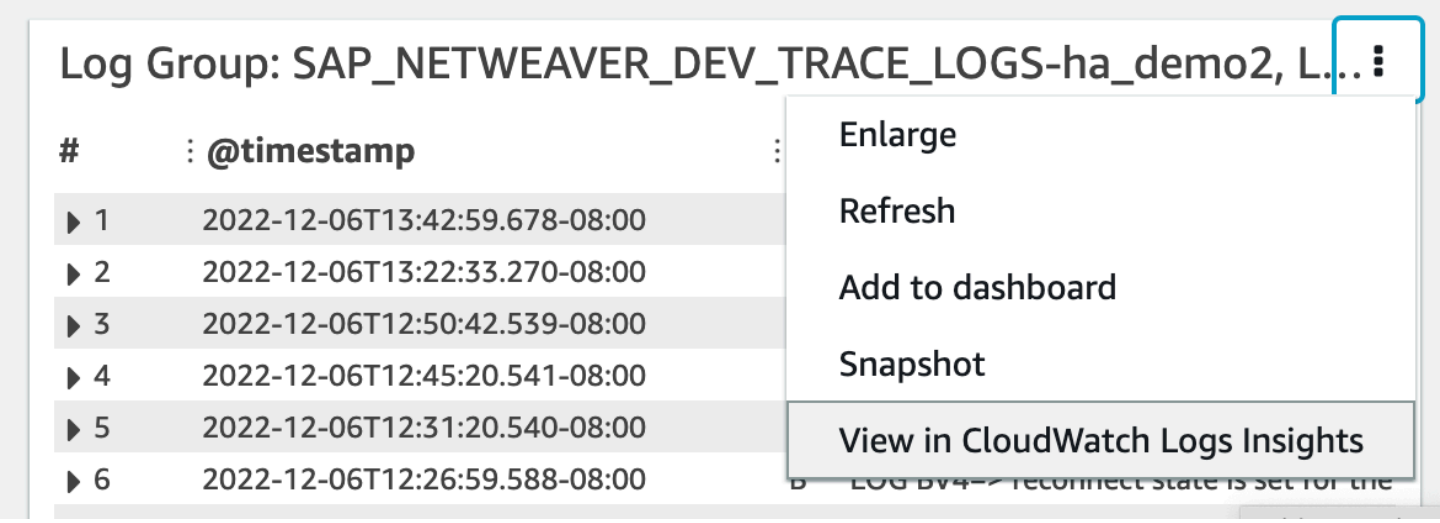


ログ

ログエントリは、問題が検出されたときに SAP NetWeaver レイヤーで発生した問題をより適切に理解するのに役立ちます。問題ダッシュボードのロググループウィジェットには、問題の具体的な発生時刻が表示されます。



ログに関する詳細情報を表示するには、右上隅にある縦に並んだ 3 つのドットを選択し、[View in CloudWatch Logs Insights] (CloudWatch Logs Insights で表示) を選択します。

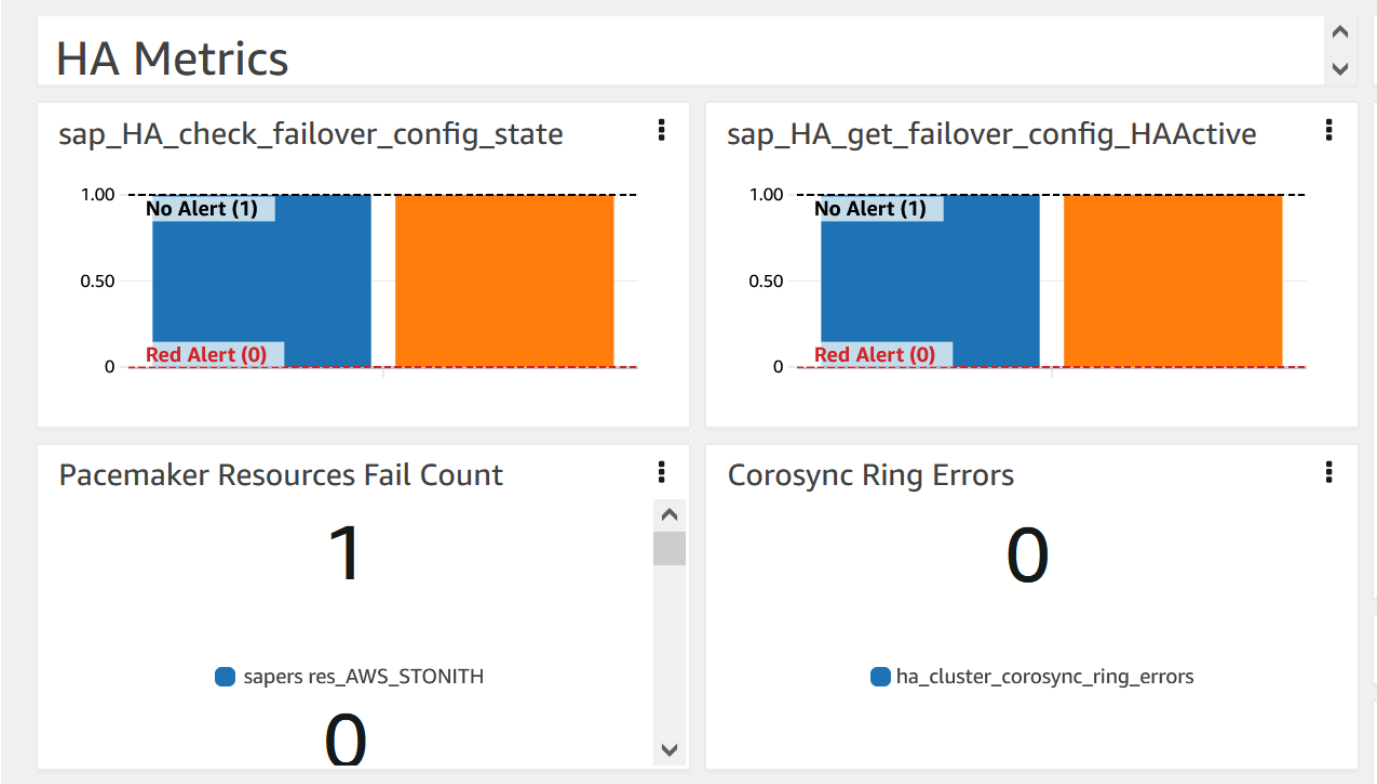


次の手順を使用して、問題ダッシュボードに表示されたメトリクスとアラームに関する詳細情報を取得します。

メトリクスとアラームに関する追加情報を取得するには

1. CloudWatch コンソールを開きます。
2. 左側のナビゲーションペインで、[Insights] (インサイト) の下にある [Application Insights] を選択します。次に、[List view] (リストビュー) タブを選択し、アプリケーションを選択します。
3. [Components] (コンポーネント) タブを選択します。次に、詳細情報を取得する SAP NetWeaver コンポーネントを選択します。

次の例は、問題ダッシュボードに表示された ha_cluster_pacemaker_fail_count メトリクスを含む [HA Metrics] (HA メトリクス) セクションを示しています。

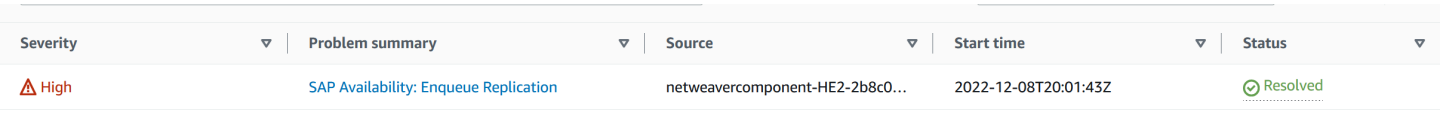


解決方法

Application Insights は、検出された問題を 1 時間ごとにモニターリングします。SAP NetWeaver ログファイルに新しい関連ログエントリがない場合、古いログエントリは解決済みとして処理されます。この問題に関連するエラー条件をすべて修正する必要があります。

sap_alerts_Shortdumps アラームについては、トランザクションコード RZ20 # R3Abap # Shortdumps を使用して CCMS アラートに移動し、SAP NetWeaver システム内のアラートを解決する必要があります。CCMS アラートの詳細については、「SAP Web サイト」を参照してください。Shortdumps ツリーのすべての CCMS アラートを解決します。SAP NetWeaver システムですべてのアラートが解決されると、CloudWatch はアラーム状態のメトリクスを報告しなくなります。

CloudWatch ログのエラーとアラームがすべて解決されると、Application Insights はエラーの検出を停止し、問題は 1 時間以内に自動的に解決されます。問題ダッシュボードに最新の問題が表示されるように、ログのエラー条件とアラームすべてを解決することをお勧めします。次の例では、「SAP Netweaver High Availability Enqueue Replication」の問題が解決されています。



SAP Netweaver 用 Application Insights のトラブルシューティング

このセクションでは、Application Insights ダッシュボードから返される一般的なエラーを解決するための手順について説明します。

60 を超えるモニターリングメトリクスを追加できない

返されたエラー: `Component cannot have more than 60 monitored metrics.`

根本原因: `The current metric limit is 60 monitor metrics per component.`

解決方法: 制限の準拠に必要のないメトリクスを削除します。

オンボーディング処理後、SAP メトリクスがダッシュボードに表示されない

根本原因: コンポーネントダッシュボードでは、データポイントを集約するために 5 分のメトリクス期間が使用されます。

解決方法: 5 分後にすべてのメトリクスがダッシュボードに表示されます。

SAP メトリクスとアラームがダッシュボードに表示されない

次の手順に従って、オンボーディング処理後に SAP メトリクスとアラームがダッシュボードに表示されない理由を特定してください。

メトリクスとアラームの問題を特定するには

1. CloudWatch コンソールを開きます。
2. 左側のナビゲーションペインで、[Insights] (インサイト) の下にある [Application Insights] を選択します。次に、[List view] (リストビュー) タブを選択し、アプリケーションを選択します。
3. [Configuration history] (設定履歴) タブを選択します。
4. メトリクスのデータポイントが欠落している場合は、`prometheus-sap_host_exporter` に関連するエラーがないかチェックしてください。
5. 前のステップでエラーが見つからなかった場合は、Linux インスタンスに接続します。High Availability デプロイでは、プライマリクラスターの Amazon EC2 インスタンスに接続します。

6. インスタンスで、次のコマンドを使用してエクスポーターが実行されていることを検証します。デフォルトのポート番号は 9680 です。別のポートを使用している場合は、9680 を使用しているポート番号に置き換えてください。

```
curl localhost:9680/metrics
```

データが返されない場合、エクスポーターは起動に失敗しています。

7. 次の 2 つのステップで WORKLOAD_SHORT_NAME に使用する正しい命名規則を見つけるには、次のコマンドを実行します。

> **Note**
>
> Application Insights は、実行中のワークロードに応じて、サービス名にサフィックス WORKLOAD_SHORT_NAME を追加します。NetWeaver Distributed、Standard、High Availability デプロイの短縮名は SAP_NWD、SAP_NWS、SAP_NWH です。

```
sudo systemctl | grep exporter
```

8. エクスポーターのサービスログにエラーがないかチェックするには、次のコマンドを実行します。

```
sudo journalctl -e --unit=prometheus-sap_host_exporter_WORKLOAD_SHORT_NAME.service
```

9. エクスポーターマネージャーのサービスログにエラーがないかチェックするには、次のコマンドを実行します。

```
sudo journalctl -e --unit=prometheus-
sap_host_exporter_manager_WORKLOAD_SHORT_NAME.service
```

> **Note**
>
> このサービスを常時起動し稼働させておく必要があります。

このコマンドでエラーが返されない場合は、次のステップに進みます。

10. エクスポーターを手動で起動するには、次のコマンドを実行します。次に、エクスポーターの出力をチェックします。

```
sudo /opt/aws/sap_host_exporter/sap_host_exporter
```

エラーをチェックしたら、エクスポーターの処理を終了できます。

根本原因: この問題の原因はいくつか考えられます。一般的な原因は、エクスポーターがアプリケーションサーバーインスタンスのいずれかに接続できなかったことです。

[Resolution] (解像度)

次の手順を使用して、エクスポーターをアプリケーションサーバーインスタンスに接続します。SAP アプリケーションインスタンスが実行中であることを確認し、SAPControl を使用してインスタンスに接続します。

エクスポーターをアプリケーションサーバーインスタンスに接続するには

1. Amazon EC2 インスタンスで、次のコマンドを実行して SAP アプリケーションが実行されていることを検証します。

```
sapcontrol -nr <App_InstNo> -function GetProcessList
```

2. 正常に機能する SAPControl 接続を確立する必要があります。SAPControl 接続が機能しない場合は、関連する SAP アプリケーションインスタンスで問題の根本原因を見つけてください。
3. SAPControl 接続の問題を修正した後にエクスポーターを手動で起動するには、次のコマンドを実行します。

```
sudo systemctl start prometheus-sap_host_exporter.service
```

4. SAPControl 接続の問題を解決できない場合は、一時的な修正として次の手順を使用してください。
   a. AWS Systems Manager コンソールを開きます。
   b. 左側のナビゲーションペインで、[State Manager] (ステートマネージャー) を選択します。
   c. [Associations] (関連付け) で、SAP NetWeaver システムの関連付けを検索します。

```
Association Name: Equal: AWS-ApplicationInsights-
SSMSAPHostExporterAssociationForCUSTOMSAPNW<SID>-1
```

d. [Association id] (関連付け ID) を選択します。

e. [Parameters] (パラメータ) タブを選択し、additionalArguments からアプリケーションサーバー番号を削除します。

f. [Apply association now] (関連付けを今すぐ適用) を選択します。

> **Note**
>
> これは一時的な修正です。コンポーネントのモニターリング設定が更新されると、インスタンスは再び追加されます。

Amazon CloudWatch Application Insights での検出された問題の表示とトラブルシューティング

このセクションのトピックでは、Application Insights で表示される検出された問題とインサイトに関する詳細情報を提供します。また、アカウントまたは設定で検出された問題の推奨される解決策も提供します。

トラブルシューティングのトピック

- CloudWatch コンソールの概要
- Application Insights の問題の概要ページ
- CloudWatch エージェントのマージ競合エラー
- アラームは作成されません。
- フィードバック
- 設定エラー

CloudWatch コンソールの概要

モニターリングされるアプリケーションに影響する問題の概要は、CloudWatch コンソールの概要ページで、CloudWatch Application Insights ペインの下に表示されます。詳細については、「Amazon CloudWatch Application Insights の開始方法」を参照してください。

CloudWatch Application Insights の概要ペインには、次が表示されます。

- 検出された問題の重大度: 高/中/低
- 問題の簡単な概要
- 問題のソース
- 問題が発生した時刻
- 問題の解決のステータス
- 影響を受けるリソースグループ

特定の問題の詳細を確認するには、[Problem Summary] (問題の概要) で、問題の説明を選択します。詳細なダッシュボードに、問題へのインサイト、関連メトリクスの異常、およびログエラーのスニペットが表示されます。インサイトが有益かどうかを選択することで、インサイトの関連度に関するフィードバックを提供できます。

新しい未設定のリソースが検出された場合は、問題の概要の説明から [Edit configuration] (設定の編集) ウィザードに移動して新しいリソースを設定します。詳細なダッシュボードの右上にある [View/edit configuration] (設定の表示/編集) を選択して、リソースグループの設定を表示または編集できます。

概要に戻るには、CloudWatch Application Insights の詳細なダッシュボードのヘッダーの横にある [Back to overview (概要に戻る)] を選択します。

Application Insights の問題の概要ページ

Application Insights の問題の概要ページ

CloudWatch Application Insights では、問題の概要ページで検出された問題に関する次の情報が提供されます。

- 問題の簡単な概要
- 問題の発生日時
- 問題の重大度: 高/中/低
- 検出された問題のステータス: 進行中/解決済み
- インサイト: 検出された問題と考えられる根本原因に関して自動生成されるインサイト
- インサイトに関するフィードバック: CloudWatch Application Insights で生成されたインサイトの有益性についてユーザーが提供したフィードバック

- 関連する監視結果: さまざまなアプリケーションコンポーネントにまたがる問題に関連するメトリクスの異常とログのエラースニペットの詳細ビュー

CloudWatch エージェントのマージ競合エラー

CloudWatch Application Insights は、顧客のインスタンスに CloudWatch エージェントをインストールして設定します。これには、メトリクスまたはログの設定を含む CloudWatch エージェント設定ファイルの作成が含まれます。お客様のインスタンスに同じメトリックスまたはログに対して異なる設定が定義された CloudWatch エージェント設定ファイルがすでに存在する場合、マージ競合が発生する可能性があります。マージ競合を解決するには、次の手順に従います。

1. システム上の CloudWatch エージェント設定ファイルを特定します。ファイルロケーションの詳細については、「CloudWatch エージェントファイルとロケーション」を参照してください。
2. 既存の CloudWatch エージェント設定ファイルから、Application Insights で使用するリソースの設定を削除してください。Application Insights サイト設定のみを使用する場合は、既存の CloudWatch エージェント設定ファイルを削除します。

アラームは作成されません。

一部のメトリクスでは、Application Insights はメトリクスの以前のデータポイントに基づいてアラームのしきい値を予測します。この予測を有効にするには、次の条件を満たす必要があります。

- 最近のデータポイント — 過去 24 時間以内のデータポイントが 100 以上ある必要があります。データポイントは連続している必要はなく、24 時間にわたって分散していてもかまいません。
- 履歴データ — 現在の日付の 15 日前から 1 日前までの期間内に少なくとも 100 個のデータポイントが必要です。データポイントは連続している必要はなく、15 日間にわたって分散していてもかまいません。

> **Note**
>
> 一部のメトリクスでは、Application Insights は前述の条件が満たされるまでアラームの作成を遅らせます。この場合、アラームのしきい値を設定するのに十分なデータポイントがメトリックスにないという設定履歴イベントが発生します。

フィードバック

フィードバック

検出された問題への自動生成されたインサイトが有益であるかどうかを判定することで、インサイトに対するフィードバックを提供できます。インサイトに対するユーザーからのフィードバックとアプリケーションの診断結果 (メトリクスの異常とログの例外) は、今後同様の問題が発生した場合の検出を向上させるために使用されます。

設定エラー

CloudWatch Application Insights は、ユーザーの設定を使用してコンポーネントのモニターリングテレメトリを作成します。Application Insights は、ユーザーのアカウントや設定で問題を検出すると、アプリケーション設定の問題を解決する方法に関する情報を [Remarks] (解説) フィールドに表示します。

次の表は、解説ごとの推奨される解決策を示しています。

解説	推奨される解決策	追加のメモ
CloudFormation のクォータにすでに達しています。	Application Insights は、すべてのアプリケーションコンポーネントについて CloudWatch エージェントのインストールと設定を管理するために、アプリケーションごとに 1 つの ClooudFormation スタックを作成します。デフォルトでは、AWS アカウントごとに 2000 のスタックを使用できます。「AWS CloudFormation 制限」をご参照ください。この問題を解決するには、CloudFormation スタックの制限を引き上げます。	該当なし

解説	推奨される解決策	追加のメモ
以下のインスタンスに SSM インスタンスロールがありません。	Application Insights でアプリケーションインスタンスに CloudWatch エージェントをインストールして設定するには、AmazonSSMManagedInstanceCore ポリシーと CloudWatchAgentServerPolicy ポリシーをインスタンスロールにアタッチする必要があります。	Application Insights は、SSM DescribeInstanceInformation API を呼び出して、SSM アクセス許可が設定されたインスタンスのリストを取得します。ロールをインスタンスにアタッチしてから SSM がインスタンスを DescribeInstanceInformation の結果に含めるまでに時間がかかります。SSM がインスタンスを結果に含めるまでは、アプリケーションの NO_SSM_INSTANCE_ROLE エラーがそのまま残ります。
新しいコンポーネントは設定を必要とする場合があります。	Application Insights は、アプリケーションのリソースグループに新しいコンポーネントがあることを検出しました。この問題を解決するには、新しいコンポーネントを適切に設定します。	該当なし

Amazon CloudWatch Application Insights でサポートされるログとメトリクス

Amazon CloudWatch Application Insights でサポートされるログとメトリクスを以下に一覧表示します。

CloudWatch Application Insights でサポートされるログは以下のとおりです。

- Microsoft Internet Information Services (IIS) のログ
- EC2 における SQL Server のエラーログ

- .NET アプリケーションのカスタムログ (Log4Net など)
- Windows ログ (システム、アプリケーション、セキュリティ)、アプリケーションとサービスログを含む Windows イベントログ
- AWS Lambda の Amazon CloudWatch Logs
- EC2 上の RDS MySQL、Aurora MySQL、MySQL のエラーログとスローログ
- EC2 上の PostgreSQL RDS および PostgreSQL の Postgresql ログ
- AWS Step Functions の Amazon CloudWatch Logs
- API Gateway REST API ステージの実行ログとアクセスログ (JSON、CSV、XML、ただし、CLF を除く)
- Prometheus JMX Exporter ログ (EMF)
- Amazon RDS の Oracle と Amazon EC2 の Oracle のアラートログとリスナーログ
- コンテナは、awslogs ログドライバーを使用して Amazon ECS コンテナから CloudWatch へのルーティングを記録します。
- コンテナは、FireLens コンテナログルーターを使用して Amazon ECS コンテナから CloudWatch へのルーティングを記録します。
- コンテナは、Container Insights を含む Fluent Bit または Fluentd ログプロセッサを使用して、Amazon EC2 で実行されている Amazon EKS または Kubernetes から CloudWatch へのルーティングを記録します。
- SAP HANA の追跡およびエラーログ
- HA Pacemaker ログ
- SAP ASE サーバーログ
- SAP ASE バックアップサーバーログ
- SAP ASE Replication サーバーログ
- SAP ASE RMA エージェントログ
- SAP ASE Fault Manager ログ
- SAP NetWeaver 開発者のトレースログ
- CloudWatch エージェント用の proctstat プラグインを使用した Windows プロセスのプロセスメトリクス
- ホストゾーンのパブリック DNS クエリログ
- Amazon Route 53 Resolver DNS クエリログ

CloudWatch Application Insights でサポートされるログクラスは以下のとおりです。

- Standard — Amazon CloudWatch Application Insights では、モニタリングを有効にするために、ロググループを CloudWatch Logs Standard ログクラスで設定する必要があります。

CloudWatch Application Insights でサポートされるアプリケーションコンポーネントのメトリクスは以下のとおりです。

- Amazon Elastic Compute Cloud (EC2)
  - CloudWatch の組み込みメトリクス
  - CloudWatch エージェントのメトリクス (Windows サーバー)
  - CloudWatch エージェントのプロセスメトリクス (Windows サーバー)
  - CloudWatch エージェントのメトリクス (Linux サーバー)
- Elastic Block Store (EBS)
- Amazon Elastic File System (Amazon EFS)
- Elastic Load Balancer (ELB)
- Application ELB
- Amazon EC2 Auto Scaling グループ
- Amazon Simple Queue Server (SQS)
- Amazon Relational Database Service (RDS)
  - RDS データベースインスタンス
  - RDS データベースクラスター
- AWS Lambda 関数
- Amazon DynamoDB テーブル
- Amazon S3 バケット
- AWS Step Functions
  - Execution-level
  - アクティビティ
  - Lambda 関数
  - サービス統合
  - Step Functions API
- API Gateway REST API ステージ
- SAP HANA
- SAP ASE

- Amazon EC2 での SAP ASE High Availability
- SAP NetWeaver
- HA Cluster
- Java
- Amazon Elastic Container Service (Amazon ECS)
  - CloudWatch の組み込みメトリクス
  - Container Insights のメトリクス
  - Container Insights Prometheus メトリクス
- AWS での Kubernetes
  - Container Insights のメトリクス
  - Container Insights Prometheus メトリクス
- Amazon FSx
- Amazon VPC
- Amazon VPC NAT ゲートウェイ
- Amazon Route 53 ヘルスチェック
- Amazon Route 53 ホストゾーン
- Amazon Route 53 Resolver エンドポイント
- AWS Network Firewall ルールグループ
- AWS Network Firewall ルールグループ関連付け
- メトリクスのデータポイント要件
  - AWS/ApplicationELB
  - AWS Autoscaling
  - AWS/EC2
  - Elastic Block Store (EBS)
  - AWS/ELB
  - AWS/RDS
  - AWS/Lambda
  - AWS/SQS
  - AWS/CWAgent
  - AWS/DynamoDB

  - AWS/S3
  - AWS/States
  - AWS/ApiGateway
  - AWS/SNS
- 推奨メトリクス
- パフォーマンスカウンタのメトリクス

Amazon Elastic Compute Cloud (EC2)

CloudWatch Application Insights でサポートされるメトリクスは次のとおりです。

メトリクス

- CloudWatch の組み込みメトリクス
- CloudWatch エージェントのメトリクス (Windows サーバー)
- CloudWatch エージェントのプロセスメトリクス (Windows サーバー)
- CloudWatch エージェントのメトリクス (Linux サーバー)

CloudWatch の組み込みメトリクス

CPUCreditBalance

CPUCreditUsage

CPUSurplusCreditBalance

CPUSurplusCreditsCharged

CPUUtilization

DiskReadBytes

DiskReadOps

DiskWriteBytes

DiskWriteOps

EBSByteBalance%

EBSIOBalance%

EBSReadBytes

EBSReadOps

EBSWriteBytes

EBSWriteOps

NetworkIn

NetworkOut

NetworkPacketsIn

NetworkPacketsOut

StatusCheckFailed

StatusCheckFailed_Instance

StatusCheckFailed_System

CloudWatch エージェントのメトリクス (Windows サーバー)

.NET CLR Exceptions # of Exceps Thrown

.NET CLR Exceptions # of Exceps Thrown/Sec

.NET CLR Exceptions # of Filters/sec

.NET CLR Exceptions # of Finallys/sec

.NET CLR Exceptions Throw to Catch Depth/sec

.NET CLR Interop # of CCWs

.NET CLR Interop # of Stubs

.NET CLR Interop # of TLB exports/sec

.NET CLR Interop # of TLB imports/sec

.NET CLR Interop # of marshaling

.NET CLR Jit % Time in Jit

.NET CLR Jit Standard Jit Failures

.NET CLR Loading % Time Loading

.NET CLR Loading Rate of Load Failures

.NET CLR LocksAndThreads Contention Rate/sec

.NET CLR LocksAndThreads Queue Length/sec

.NET CLR Memory # Total Committed Bytes

.NET CLR Memory % Time in GC

.NET CLR Networking 4.0.0.0 HttpWebRequest Average Queue Time

.NET CLR Networking 4.0.0.0 HttpWebRequests Aborted/sec

.NET CLR Networking 4.0.0.0 HttpWebRequests Failed/sec

.NET CLR Networking 4.0.0.0 HttpWebRequests Queued/sec

APP_POOL_WAS Total Worker Process Ping Failures

ASP.NET Application Restarts

ASP.NET Applications % Managed Processor Time (estimated)

ASP.NET Applications Errors Total/Sec

ASP.NET Applications Errors Unhandled During Execution/sec

ASP.NET Applications Requests in Application Queue

ASP.NET Applications Requests/Sec

ASP.NET Request Wait Time

ASP.NET Requests Queued

HTTP Service Request Queues CurrentQueueSize

LogicalDisk % Free Space

Memory % Committed Bytes In Use

Memory Available Mbytes

Memory Pages/sec

Network Interface Bytes Total/sec

Paging File % Usage

PhysicalDisk % Disk Time

PhysicalDisk Avg。Disk Queue Length

PhysicalDisk Avg。Disk sec/Read

PhysicalDisk Avg。Disk sec/Write

PhysicalDisk Disk Read Bytes/sec

PhysicalDisk Disk Reads/sec

PhysicalDisk Disk Write Bytes/sec

PhysicalDisk Disk Writes/sec

Processor % Idle Time

Processor % Interrupt Time

Processor % Processor Time

Processor % User Time

SQLServer:Access Methods Forwarded Records/sec

SQLServer:Access Methods Full Scans/sec

SQLServer:Access Methods Page Splits/sec

SQLServer:Buffer Manager Buffer cache hit ratio

SQLServer:Buffer Manager Page life expectancy

SQLServer:General Statistics Processes blocked

SQLServer:General Statistics User Connections

SQLServer:Latches Average Latch Wait Time (ms)

SQLServer:Locks Average Wait Time (ms)

SQLServer:Locks Lock Timeouts/sec

SQLServer:Locks Lock Waits/sec

SQLServer:Locks Number of Deadlocks/sec

SQLServer:Memory Manager Memory Grants Pending

SQLServer:SQL Statistics Batch Requests/sec

SQLServer:SQL Statistics SQL Compilations/sec

SQLServer:SQL Statistics SQL Re-Compilations/sec

System Processor Queue Length

TCPv4 Connections Established

TCPv6 Connections Established

W3SVC_W3WP File Cache Flushes

W3SVC_W3WP File Cache Misses

W3SVC_W3WP Requests/Sec

W3SVC_W3WP URI Cache Flushes

W3SVC_W3WP URI Cache Misses

Web Service Bytes Received/Sec

Web Service Bytes Sent/Sec

Web Service Connection attempts/sec

Web Service Current Connections

Web Service Get Requests/sec

Web Service Post Requests/sec

受信したバイト数/秒

通常のメッセージキューの長さ/秒

緊急のメッセージキューの長さ/秒

再接続の回数

未確認のメッセージキューの長さ/秒

未処理のメッセージ

送信されたメッセージ/秒

データベース更新メッセージ/秒

更新メッセージ/秒

フラッシュ/秒

保存された暗号化のチェックポイント/秒

復元された暗号化のチェックポイント/秒

復元されたレジストリのチェックポイント/秒

保存されたレジストリのチェックポイント/秒

クラスターでの API コール/秒

リソースでの API コール/秒

クラスターでの処理/秒

リソースでの処理/秒

CloudWatch エージェントのプロセスメトリクス (Windows サーバー)

プロセスメトリクスは、CloudWatch エージェント procstat プラグインを使用して収集されます。Windows ワークロードを実行する Amazon EC2 インスタンスのみがプロセスメトリクスをサポートします。

procstat cpu_time_system

procstat cpu_time_user

procstat cpu_usage

procstat memory_rss

procstat memory_vms

procstat read_bytes

procstat write_bytes

.procstat read_count

procstat write_count

CloudWatch エージェントのメトリクス (Linux サーバー)

cpu_time_active

cpu_time_guest

cpu_time_guest_nice

cpu_time_idle

cpu_time_iowait

cpu_time_irq

cpu_time_nice

cpu_time_softirq

cpu_time_steal

cpu_time_system

cpu_time_user

cpu_usage_active

cpu_usage_guest

cpu_usage_guest_nice

cpu_usage_idle

cpu_usage_iowait

cpu_usage_irq

cpu_usage_nice

cpu_usage_softirq

cpu_usage_steal

cpu_usage_system

cpu_usage_user

disk_free

disk_inodes_free

disk_inodes_used

disk_used

disk_used_percent

diskio_io_time

diskio_iops_in_progress

diskio_read_bytes

diskio_read_time

diskio_reads

diskio_write_bytes

diskio_write_time

diskio_writes

mem_active

mem_available

mem_available_percent

mem_buffered

mem_cached

mem_free

mem_inactive

mem_used

mem_used_percent

net_bytes_recv

net_bytes_sent

net_drop_in

net_drop_out

net_err_in

net_err_out

net_packets_recv

net_packets_sent

netstat_tcp_close

netstat_tcp_close_wait

netstat_tcp_closing

netstat_tcp_established

netstat_tcp_fin_wait1

netstat_tcp_fin_wait2

netstat_tcp_last_ack

netstat_tcp_listen

netstat_tcp_none

netstat_tcp_syn_recv

netstat_tcp_syn_sent

netstat_tcp_time_wait

netstat_udp_socket

processes_blocked

processes_dead

processes_idle

processes_paging

processes_running

processes_sleeping

processes_stopped

processes_total

processes_total_threads

processes_wait

processes_zombies

swap_free

swap_used

swap_used_percent

Elastic Block Store (EBS)

CloudWatch Application Insights でサポートされるメトリクスは次のとおりです。

VolumeReadBytes

VolumeWriteBytes

VolumeReadOps

VolumeWriteOps

VolumeTotalReadTime

VolumeTotalWriteTime

VolumeIdleTime

VolumeQueueLength

VolumeThroughputPercentage

VolumeConsumedReadWriteOps

BurstBalance

Amazon Elastic File System (Amazon EFS)

CloudWatch Application Insights でサポートされるメトリクスは次のとおりです。

BurstCreditBalance

PercentIOLimit

PermittedThroughput

MeteredIOBytes

TotalIOBytes

DataWriteIOBytes

DataReadIOBytes

MetadataIOBytes

ClientConnections

TimeSinceLastSync

StorageBytes

スループット

PercentageOfPermittedThroughputUtilization

ThroughputIOPS

PercentThroughputDataReadIOByte

PercentThroughputDataWriteIOBytes

PercentageOfIOPSDataReadIOBytes

PercentageOfIOPSDataWriteIOBytes

AverageDataReadIOBytesSize

AverageDataWriteIOBytesSize

Elastic Load Balancer (ELB)

CloudWatch Application Insights でサポートされるメトリクスは次のとおりです。

EstimatedALBActiveConnectionCount

EstimatedALBConsumedLCUs

EstimatedALBNewConnectionCount

EstimatedProcessedBytes

HTTPCode_Backend_4XX

HTTPCode_Backend_5XX

HealthyHostCount

RequestCount

UnHealthyHostCount

Application ELB

CloudWatch Application Insights でサポートされるメトリクスは次のとおりです。

EstimatedALBActiveConnectionCount

EstimatedALBConsumedLCUs

EstimatedALBNewConnectionCount

EstimatedProcessedBytes

HTTPCode_Backend_4XX

HTTPCode_Backend_5XX

HealthyHostCount

レイテンシー

RequestCount

SurgeQueueLength

UnHealthyHostCount

Amazon EC2 Auto Scaling グループ

CloudWatch Application Insights でサポートされるメトリクスは次のとおりです。

CPUCreditBalance

CPUCreditUsage

CPUSurplusCreditBalance

CPUSurplusCreditsCharged

CPUUtilization

DiskReadBytes

DiskReadOps

DiskWriteBytes

DiskWriteOps

EBSByteBalance%

EBSIOBalance%

EBSReadBytes

EBSReadOps

EBSWriteBytes

EBSWriteOps

NetworkIn

NetworkOut

NetworkPacketsIn

NetworkPacketsOut

StatusCheckFailed

StatusCheckFailed_Instance

StatusCheckFailed_System

Amazon Simple Queue Server (SQS)

CloudWatch Application Insights でサポートされるメトリクスは次のとおりです。

ApproximateAgeOfOldestMessage

ApproximateNumberOfMessagesDelayed

ApproximateNumberOfMessagesNotVisible

ApproximateNumberOfMessagesVisible

NumberOfEmptyReceives

NumberOfMessagesDeleted

NumberOfMessagesReceived

NumberOfMessagesSent

Amazon Relational Database Service (RDS)

CloudWatch Application Insights でサポートされるメトリクスは次のとおりです。

メトリクス

- RDS データベースインスタンス
- RDS データベースクラスター

RDS データベースインスタンス

BurstBalance

CPUCreditBalance

CPUUtilization

DatabaseConnections

DiskQueueDepth

FailedSQLServerAgentJobsCount

FreeStorageSpace

FreeableMemory

NetworkReceiveThroughput

NetworkTransmitThroughput

ReadIOPS

ReadLatency

ReadThroughput

WriteIOPS

WriteLatency

WriteThroughput

RDS データベースクラスター

ActiveTransactions

AuroraBinlogReplicaLag

AuroraReplicaLag

BackupRetentionPeriodStorageUsed

BinLogDiskUsage

BlockedTransactions

BufferCacheHitRatio

CPUUtilization

CommitLatency

CommitThroughput

DDLLatency

DDLThroughput

DMLLatency

DMLThroughput

DatabaseConnections

デッドロック

DeleteLatency

DeleteThroughput

EngineUptime

FreeLocalStorage

FreeableMemory

InsertLatency

InsertThroughput

LoginFailures

NetworkReceiveThroughput

NetworkThroughput

NetworkTransmitThroughput

クエリ

ResultSetCacheHitRatio

SelectLatency

SelectThroughput

SnapshotStorageUsed

TotalBackupStorageBilled

UpdateLatency

UpdateThroughput

VolumeBytesUsed

VolumeReadiOps

VolumeWriteIOPs

AWS Lambda 関数

CloudWatch Application Insights でサポートされるメトリクスは次のとおりです。

エラー

DeadLetterErrors

所要時間

スロットリング

IteratorAge

ProvisionedConcurrencySpilloverInvocations

Amazon DynamoDB テーブル

CloudWatch Application Insights でサポートされるメトリクスは次のとおりです。

SystemErrors

UserErrors

ConsumedReadCapacityUnits

ConsumedWriteCapacityUnits

調整イベントの読み込み

調整イベントの書き込み

TimeToLiveDeletedItemCount

条件チェックが失敗したリクエスト

TransactionConflict

ReturnedRecordsCount

PendingReplicationCount

ReplicationLatency

Amazon S3 バケット

CloudWatch Application Insights でサポートされるメトリクスは次のとおりです。

ReplicationLatency

BytesPendingReplication

OperationsPendingReplication

4xxErrors

5xxErrors

AllRequests

GetRequests

PutRequests

DeleteRequests

HeadRequests

PostRequests

SelectRequests

ListRequests

SelectScannedBytes

SelectReturnedBytes

FirstByteLatency

TotalRequestLatency

BytesDownloaded

BytesUploaded

AWS Step Functions

CloudWatch Application Insights でサポートされるメトリクスは次のとおりです。

メトリクス

- Execution-level
- アクティビティ
- Lambda 関数
- サービス統合
- Step Functions API

Execution-level

ExecutionTime

ExecutionThrottled

ExecutionsFailed

ExecutionsTimedOut

ExecutionsAborted

ExecutionsSucceeded

ExecutionsStarted

アクティビティ

ActivityRunTime

ActivityScheduleTime

ActivityTime

ActivitiesFailed

ActivitiesHeartbeatTimedOut

ActivitiesTimedOut

ActivitiesScheduled

ActivitiesSucceeded

ActivitiesStarted

Lambda 関数

LambdaFunctionRunTime

LambdaFunctionScheduleTime

LambdaFunctionTime

LambdaFunctionsFailed

LambdaFunctionsTimedOut

LambdaFunctionsScheduled

LambdaFunctionsSucceeded

LambdaFunctionsStarted

サービス統合

ServiceIntegrationRunTime

ServiceIntegrationScheduleTime

ServiceIntegrationTime

ServiceIntegrationsFailed

ServiceIntegrationsTimedOut

ServiceIntegrationsScheduled

ServiceIntegrationsSucceeded

ServiceIntegrationsStarted

Step Functions API

ThrottledEvents

ProvisionedBucketSize

ProvisionedRefillRate

ConsumedCapacity

API Gateway REST API ステージ

CloudWatch Application Insights でサポートされるメトリクスは次のとおりです。

4XXError

5XXError

IntegrationLatency

レイテンシー

CacheHitCount

CacheMissCount

SAP HANA

> **Note**
>
> CloudWatch Application Insights は、単一の SID HANA 環境のみをサポートしています。複数の HANA SID がアタッチされている場合、最初に検出された SID に対してのみモニターリングがセットアップされます。

CloudWatch Application Insights でサポートされるメトリクスは次のとおりです。

hanadb_every_service_started_status

hanadb_daemon_service_started_status

hanadb_preprocessor_service_started_status

hanadb_webdispatcher_service_started_status

hanadb_compileserver_service_started_status

hanadb_nameserver_service_started_status

hanadb_server_startup_time_variations_seconds

hanadb_level_5_alerts_count

hanadb_level_4_alerts_count

hanadb_out_of_memory_events_count

hanadb_max_trigger_read_ratio_percent

hanadb_max_trigger_write_ratio_percent

hanadb_log_switch_wait_ratio_percent

hanadb_log_switch_race_ratio_percent

hanadb_time_since_last_savepoint_seconds

hanadb_disk_usage_highlevel_percent

hanadb_max_converter_page_number_count

hanadb_long_running_savepoints_count

hanadb_failed_io_reads_count

hanadb_failed_io_writes_count

hanadb_disk_data_unused_percent

hanadb_current_allocation_limit_used_percent

hanadb_table_allocation_limit_used_percent

hanadb_host_total_physical_memory_mb

hanadb_host_physical_memory_used_mb

hanadb_host_physical_memory_free_mb

hanadb_swap_memory_free_mb

hanadb_swap_memory_used_mb

hanadb_host_allocation_limit_mb

hanadb_host_total_memory_used_mb

hanadb_host_total_peak_memory_used_mb

hanadb_host_total_allocation_limit_mb

hanadb_host_code_size_mb

hanadb_host_shared_memory_allocation_mb

hanadb_cpu_usage_percent

hanadb_cpu_user_percent

hanadb_cpu_system_percent

hanadb_cpu_waitio_percent

hanadb_cpu_busy_percent

hanadb_cpu_idle_percent

hanadb_long_delta_merge_count

hanadb_unsuccessful_delta_merge_count

hanadb_successful_delta_merge_count

hanadb_row_store_allocated_size_mb

hanadb_row_store_free_size_mb

hanadb_row_store_used_size_mb

hanadb_temporary_tables_count

hanadb_large_non_compressed_tables_count

hanadb_total_non_compressed_tables_count

hanadb_longest_running_job_seconds

hanadb_average_commit_time_milliseconds

hanadb_suspended_sql_statements_count

hanadb_plan_cache_hit_ratio_percent

hanadb_plan_cache_lookup_count

hanadb_plan_cache_hit_count

hanadb_plan_cache_total_execution_microseconds

hanadb_plan_cache_cursor_duration_microseconds

hanadb_plan_cache_preparation_microseconds

hanadb_plan_cache_evicted_count

hanadb_plan_cache_evicted_microseconds

hanadb_plan_cache_evicted_preparation_count

hanadb_plan_cache_evicted_execution_count

hanadb_plan_cache_evicted_preparation_microseconds

hanadb_plan_cache_evicted_cursor_duration_microseconds

hanadb_plan_cache_evicted_total_execution_microseconds

hanadb_plan_cache_evicted_plan_size_mb

hanadb_plan_cache_count

hanadb_plan_cache_preparation_count

hanadb_plan_cache_execution_count

hanadb_network_collision_rate

hanadb_network_receive_rate

hanadb_network_transmit_rate

hanadb_network_packet_receive_rate

hanadb_network_packet_transmit_rate

hanadb_network_transmit_error_rate

hanadb_network_receive_error_rate

hanadb_time_until_license_expires_days

hanadb_is_license_valid_status

hanadb_local_running_connections_count

hanadb_local_idle_connections_count

hanadb_remote_running_connections_count

hanadb_remote_idle_connections_count

hanadb_last_full_data_backup_age_days

hanadb_last_data_backup_age_days

hanadb_last_log_backup_age_hours

hanadb_failed_data_backup_past_7_days_count

hanadb_failed_log_backup_past_7_days_count

hanadb_oldest_backup_in_catalog_age_days

hanadb_backup_catalog_size_mb

hanadb_hsr_replication_status

hanadb_hsr_log_shipping_delay_seconds

hanadb_hsr_secondary_failover_count

hanadb_hsr_secondary_reconnect_count

hanadb_hsr_async_buffer_used_mb

hanadb_hsr_secondary_active_status

hanadb_handle_count

hanadb_ping_time_milliseconds

hanadb_connection_count

hanadb_internal_connection_count

hanadb_external_connection_count

hanadb_idle_connection_count

hanadb_transaction_count

hanadb_internal_transaction_count

hanadb_external_transaction_count

hanadb_user_transaction_count

hanadb_blocked_transaction_count

hanadb_statement_count

hanadb_active_commit_id_range_count

hanadb_mvcc_version_count

hanadb_pending_session_count

hanadb_record_lock_count

hanadb_read_count

hanadb_write_count

hanadb_merge_count

hanadb_unload_count

hanadb_active_thread_count

hanadb_waiting_thread_count

hanadb_total_thread_count

hanadb_active_sql_executor_count

hanadb_waiting_sql_executor_count

hanadb_total_sql_executor_count

hanadb_data_write_size_mb

hanadb_data_write_time_milliseconds

hanadb_log_write_size_mb

hanadb_log_write_time_milliseconds

hanadb_data_read_size_mb

hanadb_data_read_time_milliseconds

hanadb_log_read_size_mb

hanadb_log_read_time_milliseconds

hanadb_data_backup_write_size_mb

hanadb_data_backup_write_time_milliseconds

hanadb_log_backup_write_size_mb

hanadb_log_backup_write_time_milliseconds

hanadb_mutex_collision_count

hanadb_read_write_lock_collision_count

hanadb_admission_control_admit_count

hanadb_admission_control_reject_count

hanadb_admission_control_queue_size_mb

hanadb_admission_control_wait_time_milliseconds

SAP ASE

CloudWatch Application Insights でサポートされるメトリクスは次のとおりです。

asedb_database_availability

asedb_trunc_log_on_chkpt_enabled

asedb_last_db_backup_age_in_days

asedb_last_transaction_log_backup_age_in_hours

asedb_suspected_database

asedb_db_space_usage_percent

asedb_db_log_space_usage_percent

asedb_locked_login

asedb_has_mixed_log_and_data

asedb_runtime_for_open_transactions

asedb_data_cache_hit_ratio

asedb_data_cache_usage

asedb_sql_cache_hit_ratio

asedb_cache_usage

asedb_run_queue_length

asedb_number_of_rollbacks

asedb_number_of_commits

asedb_number_of_transactions

asedb_outstanding_disk_io

asedb_percent_io_busy

asedb_percent_system_busy

asedb_percent_locks_active

asedb_scheduled_jobs_failed_percent

asedb_user_connections_percent

asedb_query_logical_reads

asedb_query_physical_reads

asedb_query_cpu_time

asedb_query_memory_usage

Amazon EC2 での SAP ASE High Availability

CloudWatch Application Insights でサポートされるメトリクスは次のとおりです。

asedb_ha_replication_state

asedb_ha_replication_mode

asedb_ha_replication_latency_in_minutes

SAP NetWeaver

CloudWatch Application Insights でサポートされるメトリクスは次のとおりです。

メトリクス	説明
sap_alerts_ResponseTime	CCMS (RZ20)>R3Services>Dialog>ResponseTime からの SAP 応答時間アラート。
sap_alerts_ResponseTimeDialog	CCMS (RZ20)>R3Services>Dialog>ResponseTimeDialog からの SAP 応答時間ダイアログアラート。
sap_alerts_ResponseTimeDialogRFC	CCMS (RZ20)>R3Services> Dialog>ResponseTimeDialogRFC からの SAP 応答時間アラート。
sap_alerts_DBRequestTime	CCMS (RZ20)>R3Services>Dialog>DBRequestTime からの SAP 応答時間アラート。
sap_alerts_FrontendResponseTime	CCMS (RZ20)>R3Services > Dialog>FrontEndResponseTime からの SAP 応答時間アラート。
sap_alerts_Database	SAP システムが、データベース関連のエラーをログに記録しました。SM21 または CCMS (RZ20)>R3Syslog>Database からのアラート。
sap_alerts_QueueTime	CCMS (RZ20)>R3Services>Dialog>QueueTime からの SAP キュータイムアラート。
sap_alerts_AbortedJobs	SAP システムのバックグラウンドジョブが失敗しました。(RZ20)>R3Services > Background>AbortedJobs からのアラート。
sap_alerts_BasisSystem	SAP システムがシステムレベルのエラーをログに記録しました。SM21 または CCMS (RZ20)>R3Syslog>BasisSystem からのアラート。

メトリクス	説明
sap_alerts_Security	SAP システムは、セキュリティ関連のメッセージを口グに記録しました。SM21 または CCMS (RZ20)>R3Syslog>Security からのアラート。
sap_alerts_System	SAP システムは、セキュリティまたは監査関連のメッセージを口グに記録しました。SM21 または CCMS (RZ20)>Security>System からのアラート。
sap_alerts_LongRunners	SAP システムには実行時間の長いプログラムがあります。CCMS (RZ20)>R3Services > Dialog>LongRunners からのアラート。
sap_alerts_SqlError	SAP データベースのクライアントレイヤーのエラーログが存在します。CCMS(RZ20)>DatabaseClient>AbapSql>SqlError からのアラート。
sap_alerts_State	CCMS (RZ20)>OS Collector>State からの状態アラート。
sap_alerts_Shortdumps	ST22 および CCMS (RZ20)>R3Abap>Shortdumps からのショートダンプアラート。
sap_alerts_Availability	SM21、SM50、SM51、SM66、および CCMS (RZ20)>InstanceAsTask>Availability からの SAP アプリケーションサーバーインスタンスの可用性アラート。
sap_dispatcher_queue_high	SAPControl Web サービス関数 `GetQueueStatistic` は、ディスパッチャキューの上限数を提供します。

メトリクス	説明
sap_dispatcher_queue_max	SAPControl Web サービス関数 GetQueueStatistic は、ディスパッチャキューの最大数を提供します。
sap_dispatcher_queue_now	SAPControl Web サービス関数 GetQueueStatistic は、ディスパッチャキューの現在の数を提供します。
sap_dispatcher_queue_reads	SAPControl Web サービス関数 GetQueueStatistic は、ディスパッチャキューの読み込み回数を提供します。
sap_dispatcher_queue_writes	SAPControl Web サービス関数 GetQueueStatistic は、ディスパッチャキューの書き込み回数を提供します。
sap_enqueue_server_arguments_high	SAPControl Web サービス関数 EnqGetStatistic は、高い値のエンキュー引数を提供します。
sap_enqueue_server_arguments_max	SAPControl Web サービス関数 EnqGetStatistic は、最大値のエンキュー引数を提供します。
sap_enqueue_server_arguments_now	SAPControl Web サービス関数 EnqGetStatistic は、現在の値のエンキュー引数を提供します。
sap_enqueue_server_arguments_state	SAPControl Web サービス関数 EnqGetStatistic は、エンキュー引数の状態を提供します。
sap_enqueue_server_backup_requests	SAPControl Web サービス関数 EnqGetStatistic は、エンキューのバックアップリクエスト数を提供します。

メトリクス	説明
sap_enqueue_server_cleanup_requests	SAPControl Web サービス関数 `EnqGetStatistic` は、エンキューのクリーンアップリクエスト数を提供します。
sap_enqueue_server_dequeue_all_requests	SAPControl Web サービス関数 `EnqGetStatistic` は、すべてのデキューリクエスト数を提供します。
sap_enqueue_server_dequeue_errors	SAPControl Web サービス関数 `EnqGetStatistic` は、デキューエラー数を提供します。
sap_enqueue_server_dequeue_requests	SAPControl Web サービス関数 `EnqGetStatistic` は、デキューリクエスト数を提供します。
sap_enqueue_server_enqueue_errors	SAPControl Web サービス関数 `EnqGetStatistic` は、エンキューエラー数を提供します。
sap_enqueue_server_enqueue_rejects	SAPControl Web サービス関数 `EnqGetStatistic` は、エンキューが拒否された回数を提供します。
sap_enqueue_server_enqueue_requests	SAPControl Web サービス関数 `EnqGetStatistic` は、エンキューリクエスト数を提供します。
sap_enqueue_server_lock_time	SAPControl Web サービス関数 `EnqGetStatistic` は、エンキューロックの時刻を提供します。
sap_enqueue_server_lock_wait_time	SAPControl Web サービス関数 `EnqGetStatistic` は、エンキューロックの待機時間を提供します。

メトリクス	説明
sap_enqueue_server_locks_high	SAPControl Web サービス関数 EnqGetStatistic は、エンキューロックの高い数値の数を提供します。
sap_enqueue_server_locks_max	SAPControl Web サービス関数 EnqGetStatistic は、エンキューロックの最大数を提供します。
sap_enqueue_server_locks_now	SAPControl Web サービス関数 EnqGetStatistic は、エンキューロックの現在の数を提供します。
sap_enqueue_server_locks_state	SAPControl Web サービス関数 EnqGetStatistic は、エンキューロックの状態を提供します。
sap_enqueue_server_owner_high	SAPControl Web サービス関数 EnqGetStatistic は、エンキュー所有者の高い数値の数を提供します。
sap_enqueue_server_owner_max	SAPControl Web サービス関数 EnqGetStatistic は、エンキュー所有者の最大数を提供します。
sap_enqueue_server_owner_now	SAPControl Web サービス関数 EnqGetStatistic は、エンキュー所有者の現在の数を提供します。
sap_enqueue_server_owner_state	SAPControl Web サービス関数 EnqGetStatistic は、エンキュー所有者の状態を提供します。
sap_enqueue_server_replication_state	SAPControl Web サービス関数 EnqGetStatistic は、エンキューのレプリケーション状態ステータスを提供します。

メトリクス	説明
sap_enqueue_server_reporting_requests	SAPControl Web サービス関数 `EnqGetStatistic` は、レポートリクエストのステータスを提供します。
sap_enqueue_server_server_time	SAPControl Web サービス関数 `EnqGetStatistic` は、エンキューサーバー時間を提供します。
sap_HA_check_failover_config_state	SAPControl Web サービス関数 `HACheckFailoverConfig` は、SAP High Availability のステータスを提供します。
sap_HA_get_failover_config_HAActive	SAPControl Web サービス関数 `HAGetFailoverConfig` は、SAP High Availability クラスターの設定とステータスを提供します。
sap_start_service_processes	SAPControl Web サービス関数 `GetProcessList` は、disp+work、IGS、gwrd、icman、メッセージサーバー、およびエンキューサーバープロセスのステータスを提供します。

HA Cluster

CloudWatch Application Insights でサポートされるメトリクスは次のとおりです。

ha_cluster_pacemaker_stonith_enabled

ha_cluster_corosync_quorate

hanadb_webdispatcher_service_started_status

ha_cluster_pacemaker_nodes

ha_cluster_corosync_ring_errors

ha_cluster_pacemaker_fail_count

Java

CloudWatch Application Insights でサポートされるメトリクスは次のとおりです。

java_lang_memory_heapmemoryusage_used

java_lang_memory_heapmemoryusage_committed

java_lang_operatingsystem_openfiledescriptorcount

java_lang_operatingsystem_maxfiledescriptorcount

java_lang_operatingsystem_freephysicalmemorysize

java_lang_operatingsystem_freeswapspacesize

java_lang_threading_threadcount

java_lang_threading_daemonthreadcount

java_lang_classloading_loadedclasscount

java_lang_garbagecollector_collectiontime_copy

java_lang_garbagecollector_collectiontime_ps_scavenge

java_lang_garbagecollector_collectiontime_parnew

java_lang_garbagecollector_collectiontime_marksweepcompact

java_lang_garbagecollector_collectiontime_ps_marksweep

java_lang_garbagecollector_collectiontime_concurrentmarksweep

java_lang_garbagecollector_collectiontime_g1_young_generation

java_lang_garbagecollector_collectiontime_g1_old_generation

java_lang_garbagecollector_collectiontime_g1_mixed_generation

java_lang_operatingsystem_committedvirtualmemorysize

Amazon Elastic Container Service (Amazon ECS)

CloudWatch Application Insights でサポートされるメトリクスは次のとおりです。

メトリクス

- CloudWatch の組み込みメトリクス
- Container Insights のメトリクス
- Container Insights Prometheus メトリクス

CloudWatch の組み込みメトリクス

CPUReservation

CPUUtilization

MemoryReservation

MemoryUtilization

GPUReservation

Container Insights のメトリクス

ContainerInstanceCount

CpuUtilized

CpuReserved

DeploymentCount

DesiredTaskCount

MemoryUtilized

MemoryReserved

NetworkRxBytes

NetworkTxBytes

PendingTaskCount

RunningTaskCount

ServiceCount

StorageReadBytes

StorageWriteBytes

TaskCount

TaskSetCount

instance_cpu_limit

instance_cpu_reserved_capacity

instance_cpu_usage_total

instance_cpu_utilization

instance_filesystem_utilization

instance_memory_limit

instance_memory_reserved_capacity

instance_memory_utilization

instance_memory_working_set

instance_network_total_bytes

instance_number_of_running_tasks

Container Insights Prometheus メトリクス

Java JMX メトリクス

java_lang_memory_heapmemoryusage_used

java_lang_memory_heapmemoryusage_committed

java_lang_operatingsystem_openfiledescriptorcount

java_lang_operatingsystem_maxfiledescriptorcount

java_lang_operatingsystem_freephysicalmemorysize

java_lang_operatingsystem_freeswapspacesize

java_lang_threading_threadcount

java_lang_classloading_loadedclasscount

java_lang_threading_daemonthreadcount

java_lang_garbagecollector_collectiontime_copy

java_lang_garbagecollector_collectiontime_ps_scavenge

java_lang_garbagecollector_collectiontime_parnew

java_lang_garbagecollector_collectiontime_marksweepcompact

java_lang_garbagecollector_collectiontime_ps_marksweep

java_lang_garbagecollector_collectiontime_concurrentmarksweep

java_lang_garbagecollector_collectiontime_g1_young_generation

java_lang_garbagecollector_collectiontime_g1_old_generation

java_lang_garbagecollector_collectiontime_g1_mixed_generation

java_lang_operatingsystem_committedvirtualmemorysize

AWS での Kubernetes

CloudWatch Application Insights でサポートされるメトリクスは次のとおりです。

メトリクス

- Container Insights のメトリクス
- Container Insights Prometheus メトリクス

Container Insights のメトリクス

cluster_failed_node_count

cluster_node_count

namespace_number_of_running_pods

node_cpu_limit

node_cpu_reserved_capacity

node_cpu_usage_total

node_cpu_utilization

node_filesystem_utilization

node_memory_limit

node_memory_reserved_capacity

node_memory_utilization

node_memory_working_set

node_network_total_bytes

node_number_of_running_containers

node_number_of_running_pods

pod_cpu_reserved_capacity

pod_cpu_utilization

pod_cpu_utilization_over_pod_limit

pod_memory_reserved_capacity

pod_memory_utilization

pod_memory_utilization_over_pod_limit

pod_network_rx_bytes

pod_network_tx_bytes

service_number_of_running_pods

Container Insights Prometheus メトリクス

Java JMX メトリクス

java_lang_memory_heapmemoryusage_used

java_lang_memory_heapmemoryusage_committed

java_lang_operatingsystem_openfiledescriptorcount

java_lang_operatingsystem_maxfiledescriptorcount

java_lang_operatingsystem_freephysicalmemorysize

java_lang_operatingsystem_freeswapspacesize

java_lang_threading_threadcount

java_lang_classloading_loadedclasscount

java_lang_threading_daemonthreadcount

java_lang_garbagecollector_collectiontime_copy

java_lang_garbagecollector_collectiontime_ps_scavenge

java_lang_garbagecollector_collectiontime_parnew

java_lang_garbagecollector_collectiontime_marksweepcompact

java_lang_garbagecollector_collectiontime_ps_marksweep

java_lang_garbagecollector_collectiontime_concurrentmarksweep

java_lang_garbagecollector_collectiontime_g1_young_generation

java_lang_garbagecollector_collectiontime_g1_old_generation

java_lang_garbagecollector_collectiontime_g1_mixed_generation

java_lang_operatingsystem_committedvirtualmemorysize

Amazon FSx

CloudWatch Application Insights でサポートされるメトリクスは次のとおりです。

DataReadBytes

DataWriteBytes

DataReadOperations

DataWriteOperations

MetadataOperations

FreeStorageCapacity

FreeDataStorageCapacity

LogicalDiskUsage

PhysicalDiskUsage

Amazon VPC

CloudWatch Application Insights でサポートされるメトリクスは次のとおりです。

NetworkAddressUsage

NetworkAddressUsagePeered

VPCFirewallQueryVolume

Amazon VPC NAT ゲートウェイ

CloudWatch Application Insights でサポートされるメトリクスは次のとおりです。

ErrorPortAllocation

IdleTimeoutCount

Amazon Route 53 ヘルスチェック

CloudWatch Application Insights でサポートされるメトリクスは次のとおりです。

ChildHealthCheckHealthyCount

ConnectionTime

HealthCheckPercentageHealthy

HealthCheckStatus

SSLHandshakeTime

TimeToFirstByte

Amazon Route 53 ホストゾーン

CloudWatch Application Insights でサポートされるメトリクスは次のとおりです。

DNSQueries

DNSSECInternalFailure

DNSSECKeySigningKeysNeedingAction

DNSSECKeySigningKeyMaxNeedingActionAge

DNSSECKeySigningKeyAge

Amazon Route 53 Resolver エンドポイント

CloudWatch Application Insights でサポートされるメトリクスは次のとおりです。

EndpointHealthyENICount

EndpointUnHealthyENICount

InboundQueryVolume

OutboundQueryVolume

OutboundQueryAggregateVolume

AWS Network Firewall ルールグループ

CloudWatch Application Insights でサポートされるメトリクスは次のとおりです。

FirewallRuleGroupQueryVolume

AWS Network Firewall ルールグループ関連付け

CloudWatch Application Insights でサポートされるメトリクスは次のとおりです。

FirewallRuleGroupVpcQueryVolume

メトリクスのデータポイント要件

アラームを発生させるデフォルトのしきい値が明確でないメトリクスの場合、Application Insights は、アラームを発生させるしきい値を適切に予測できるだけのデータポイント数が得られるまで待機します。アラームが作成される前に CloudWatch Application Insights で確認するメトリクスのデータポイント要件は次のとおりです。

- メトリクスのデータポイントが、過去 15 日から 2 日までの間に 100 個以上あること。
- メトリクスのデータポイントが、最終日から 100 個以上あること。

次のメトリクスは、上記のデータポイント要件に従います。CloudWatch エージェントのメトリクスは、アラームの作成に最大 1 時間を必要とします。

メトリクス

- AWS/ApplicationELB
- AWS Autoscaling
- AWS/EC2
- Elastic Block Store (EBS)
- AWS/ELB
- AWS/RDS
- AWS/Lambda
- AWS/SQS
- AWS/CWAgent
- AWS/DynamoDB
- AWS/S3
- AWS/States
- AWS/ApiGateway
- AWS/SNS

AWS/ApplicationELB

ActiveConnectionCount

ConsumedLCUs

HTTPCode_ELB_4XX_Count

HTTPCode_Target_2XX_Count

HTTPCode_Target_3XX_Count

HTTPCode_Target_4XX_Count

HTTPCode_Target_5XX_Count

NewConnectionCount

ProcessedBytes

TargetResponseTime

UnHealthyHostCount

AWS Autoscaling

GroupDesiredCapacity

GroupInServiceInstances

GroupMaxSize

GroupMinSize

GroupPendingInstances

GroupStandbyInstances

GroupTerminatingInstances

GroupTotalInstances

AWS/EC2

CPUCreditBalance

CPUCreditUsage

CPUSurplusCreditBalance

CPUSurplusCreditsCharged

CPUUtilization

DiskReadBytes

DiskReadOps

DiskWriteBytes

DiskWriteOps

EBSByteBalance%

EBSIOBalance%

EBSReadBytes

EBSReadOps

EBSWriteBytes

EBSWriteOps

NetworkIn

NetworkOut

NetworkPacketsIn

NetworkPacketsOut

Elastic Block Store (EBS)

VolumeReadBytes

VolumeWriteBytes

VolumeReadOps

VolumeWriteOps

VolumeTotalReadTime

VolumeTotalWriteTime

VolumeIdleTime

VolumeQueueLength

VolumeThroughputPercentage

VolumeConsumedReadWriteOps

BurstBalance

AWS/ELB

EstimatedALBActiveConnectionCount

EstimatedALBConsumedLCUs

EstimatedALBNewConnectionCount

EstimatedProcessedBytes

HTTPCode_Backend_4XX

HTTPCode_Backend_5XX

HealthyHostCount

レイテンシー

RequestCount

SurgeQueueLength

UnHealthyHostCount

AWS/RDS

ActiveTransactions

AuroraBinlogReplicaLag

AuroraReplicaLag

BackupRetentionPeriodStorageUsed

BinLogDiskUsage

BlockedTransactions

CPUCreditBalance

CommitLatency

CommitThroughput

DDLLatency

DDLThroughput

DMLLatency

DMLThroughput

DatabaseConnections

デッドロック

DeleteLatency

DeleteThroughput

DiskQueueDepth

EngineUptime

FreeLocalStorage

FreeStorageSpace

FreeableMemory

InsertLatency

InsertThroughput

LoginFailures

NetworkReceiveThroughput

NetworkThroughput

NetworkTransmitThroughput

クエリ

ReadIOPS

ReadThroughput

SelectLatency

SelectThroughput

SnapshotStorageUsed

TotalBackupStorageBilled

UpdateLatency

UpdateThroughput

VolumeBytesUsed

VolumeReadiOps

VolumeWriteIOPs

WriteIOPS

WriteThroughput

AWS/Lambda

エラー

DeadLetterErrors

所要時間

スロットリング

IteratorAge

ProvisionedConcurrencySpilloverInvocations

AWS/SQS

ApproximateAgeOfOldestMessage

ApproximateNumberOfMessagesDelayed

ApproximateNumberOfMessagesNotVisible

ApproximateNumberOfMessagesVisible

NumberOfEmptyReceives

NumberOfMessagesDeleted

NumberOfMessagesReceived

NumberOfMessagesSent

AWS/CWAgent

LogicalDisk % Free Space

Memory % Committed Bytes In Use

Memory Available Mbytes

Network Interface Bytes Total/sec

Paging File % Usage

PhysicalDisk % Disk Time

PhysicalDisk Avg。Disk sec/Read

PhysicalDisk Avg。Disk sec/Write

PhysicalDisk Disk Read Bytes/sec

PhysicalDisk Disk Reads/sec

PhysicalDisk Disk Write Bytes/sec

PhysicalDisk Disk Writes/sec

Processor % Idle Time

Processor % Interrupt Time

Processor % Processor Time

Processor % User Time

SQLServer:Access Methods Forwarded Records/sec

SQLServer:Access Methods Page Splits/sec

SQLServer:Buffer Manager Buffer cache hit ratio

SQLServer:Buffer Manager Page life expectancy

SQLServer:Database Replica File Bytes Received/sec

SQLServer:Database Replica Log Bytes Received/sec

SQLServer:Database Replica Log remaining for undo

SQLServer:Database Replica Log Send Queue

SQLServer:Database Replica Mirrored Write Transaction/sec

SQLServer:Database Replica Recovery Queue

SQLServer:Database Replica Redo Bytes Remaining

SQLServer:Database Replica Redone Bytes/sec

SQLServer:Database Replica Total Log requiring undo

SQLServer:Database Replica Transaction Delay

SQLServer:General Statistics Processes blocked

SQLServer:SQL Statistics Batch Requests/sec

SQLServer:SQL Statistics SQL Compilations/sec

SQLServer:SQL Statistics SQL Re-Compilations/sec

System Processor Queue Length

TCPv4 Connections Established

TCPv6 Connections Established

AWS/DynamoDB

ConsumedReadCapacityUnits

ConsumedWriteCapacityUnits

調整イベントの読み込み

調整イベントの書き込み

TimeToLiveDeletedItemCount

条件チェックが失敗したリクエスト

TransactionConflict

ReturnedRecordsCount

PendingReplicationCount

ReplicationLatency

AWS/S3

ReplicationLatency

BytesPendingReplication

OperationsPendingReplication

4xxErrors

5xxErrors

AllRequests

GetRequests

PutRequests

DeleteRequests

HeadRequests

PostRequests

SelectRequests

ListRequests

SelectScannedBytes

SelectReturnedBytes

FirstByteLatency

TotalRequestLatency

BytesDownloaded

BytesUploaded

AWS/States

ActivitiesScheduled

ActivitiesStarted

ActivitiesSucceeded

ActivityScheduleTime

ActivityRuntime

ActivityTime

LambdaFunctionsScheduled

LambdaFunctionsStarted

LambdaFunctionsSucceeded

LambdaFunctionScheduleTime

LambdaFunctionRuntime

LambdaFunctionTime

ServiceIntegrationsScheduled

ServiceIntegrationsStarted

ServiceIntegrationsSucceeded

ServiceIntegrationScheduleTime

ServiceIntegrationRuntime

ServiceIntegrationTime

ProvisionedRefillRate

ProvisionedBucketSize

ConsumedCapacity

ThrottledEvents

AWS/ApiGateway

4XXError

IntegrationLatency

レイテンシー

DataProcessed

CacheHitCount

CacheMissCount

AWS/SNS

NumberOfNotificationsDelivered

NumberOfMessagesPublished

NumberOfNotificationsFailed

NumberOfNotificationsFilteredOut

NumberOfNotificationsFilteredOut-InvalidAttributes

NumberOfNotificationsFilteredOut-NoMessageAttributes

NumberOfNotificationsRedrivenToDlq

NumberOfNotificationsFailedToRedriveToDlq

SMSSuccessRate

推奨メトリクス

次の表は、コンポーネントタイプ別の推奨されるメトリクスの一覧です。

コンポーネントタイプ	ワークロードタイプ	推奨されるメトリクス
EC2 インスタンス (Windows サーバー)	Default/Custom	CPUUtilization StatusCheckFailed Processor % Processor Time Memory % Committed Bytes In Use LogicalDisk % Free Space Memory Available Mbytes
	アクティブディレクトリ	CPUUtilization StatusCheckFailed Processor % Processor Time Memory % Committed Bytes In Use Memory Available Mbytes Database ==> Instances Database Cache % Hit DirectoryServices DRA Pending Replication Operations DirectoryServices DRA Pending Replication Synchronizations

コンポーネントタイプ	ワークロードタイプ	推奨されるメトリクス
		DNS Recursive Query Failure/sec LogicalDisk Avg。Disk Queue Length
	Java アプリケーション	CPUUtilization StatusCheckFailed Processor % Processor Time Memory % Committed Bytes In Use Memory Available Mbytes java_lang_threading_threadcount java_lang_classloading_loadedclasscount java_lang_memory_heapmemoryusage_used java_lang_memory_heapmemoryusage_committed java_lang_operatingsystem_freephysicalmemorysize java_lang_operatingsystem_freeswapspacesize

コンポーネントタイプ	ワークロードタイプ	推奨されるメトリクス
	マイクロソフト IIS/.NET Web フロントエンド	CPUUtilization StatusCheckFailed Processor % Processor Time Memory % Committed Bytes In Use Memory Available Mbytes .NET CLR Exceptions # of Exceps Thrown/Sec .NET CLR Memory # Total Committed Bytes .NET CLR Memory % Time in GC ASP.NET Applications Requests in Application Queue ASP.NET Requests Queued ASP.NET Application Restarts

コンポーネントタイプ	ワークロードタイプ	推奨されるメトリクス
	Microsoft SQL Server Database Tier	CPUUtilization StatusCheckFailed Processor % Processor Time Memory % Committed Bytes In Use Memory Available Mbytes Paging File % Usage System Processor Queue Length Network Interface Bytes Total/sec PhysicalDisk % Disk Time SQLServer:Buffer Manager Buffer cache hit ratio SQLServer:Buffer Manager Page Life Expectancy SQLServer:General Statistics Processes Blocked SQLServer:General Statistics User Connections SQLServer:Locks Number of Deadlocks/Sec SQLServer:SQL Statistics Batch Requests/Sec

コンポーネントタイプ	ワークロードタイプ	推奨されるメトリクス
	MySQL	CPUUtilization StatusCheckFailed Processor % Processor Time Memory % Committed Bytes In Use LogicalDisk % Free Space Memory Available Mbytes
	.NET workerpool/Mid-Tier	CPUUtilization StatusCheckFailed Processor % Processor Time Memory % Committed Bytes In Use Memory Available Mbytes .NET CLR Exceptions # of Exceps Thrown/Sec .NET CLR Memory # Total Committed Bytes .NET CLR Memory % Time in GC

コンポーネントタイプ	ワークロードタイプ	推奨されるメトリクス
	.NET Core Tier	CPUUtilization StatusCheckFailed Processor % Processor Time Memory % Committed Bytes In Use Memory Available Mbytes
	Oracle	CPUUtilization StatusCheckFailed Processor % Processor Time Memory % Committed Bytes In Use LogicalDisk % Free Space Memory Available Mbytes
	Postgres	CPUUtilization StatusCheckFailed Processor % Processor Time Memory % Committed Bytes In Use LogicalDisk % Free Space Memory Available Mbytes

コンポーネントタイプ	ワークロードタイプ	推奨されるメトリクス
	SharePoint	CPUUtilization StatusCheckFailed Processor % Processor Time Memory % Committed Bytes In Use Memory Available Mbytes ASP.NET Applications Cache API trims ASP.NET Requests Rejected ASP.NET Worker Process Restarts Memory Pages/sec SharePoint Publishing Cache Publishing cache flushes / second SharePoint Foundation Executing Time/Page Request SharePoint Disk-Based Cache Total number of cache compactions SharePoint Disk-Based Cache Blob cache hit ratio SharePoint Disk-Based Cache Blob Cache fill ratio

コンポーネントタイプ	ワークロードタイプ	推奨されるメトリクス
		SharePoint Disk-Based Cache Blob cache flushes / second ASP.NET Requests Queued ASP.NET Applications Requests in Application Queue ASP.NET Application Restarts LogicalDisk Avg。Disk sec/ Write LogicalDisk Avg。Disk sec/ Read Processor % Interrupt Time
EC2 インスタンス (Linux サーバー)	Default/Custom	CPUUtilization StatusCheckFailed disk_used_percent mem_used_percent

コンポーネントタイプ	ワークロードタイプ	推奨されるメトリクス
	Java アプリケーション	CPUUtilization StatusCheckFailed disk_used_percent mem_used_percent java_lang_threading_threadcount java_lang_classloading_loadedclasscount java_lang_memory_heapmemoryusage_used java_lang_memory_heapmemoryusage_committed java_lang_operatingsystem_freephysicalmemorysize java_lang_operatingsystem_freeswapspacesize
	.NET Core Tier または SQL Server Database Tier	CPUUtilization StatusCheckFailed disk_used_percent mem_used_percent

コンポーネントタイプ	ワークロードタイプ	推奨されるメトリクス
	Oracle	CPUUtilization StatusCheckFailed disk_used_percent mem_used_percent
	Postgres	CPUUtilization StatusCheckFailed disk_used_percent mem_used_percent

コンポーネントタイプ	ワークロードタイプ	推奨されるメトリクス
EC2 インスタンスグループ	SAP HANA マルチノードまたはシングルノード	• hanadb_server_startup_time_variations_seconds • hanadb_level_5_alerts_count • hanadb_level_4_alerts_count • hanadb_out_of_memory_events_count • hanadb_max_trigger_read_ratio_percent • hanadb_max_trigger_write_ratio_percent • hanadb_log_switch_race_ratio_percent • hanadb_time_since_last_savepoint_seconds • hanadb_disk_usage_highlevel_percent • hanadb_current_allocation_limit_used_percent • hanadb_table_allocation_limit_used_percent • hanadb_cpu_usage_percent • hanadb_plan_cache_hit_ratio_percent • hanadb_last_data_backup_age_days

コンポーネントタイプ	ワークロードタイプ	推奨されるメトリクス
EBS ボリューム	すべて	VolumeReadBytes VolumeWriteBytes VolumeReadOps VolumeWriteOps VolumeQueueLength VolumeThroughputPercentage VolumeConsumedReadWriteOps BurstBalance
Classic ELB	すべて	HTTPCode_Backend_4XX HTTPCode_Backend_5XX レイテンシー SurgeQueueLength UnHealthyHostCount
Application ELB	すべて	HTTPCode_Target_4XX_Count HTTPCode_Target_5XX_Count TargetResponseTime UnHealthyHostCount

コンポーネントタイプ	ワークロードタイプ	推奨されるメトリクス
RDS データベースインスタンス	すべて	CPUUtilization ReadLatency WriteLatency BurstBalance FailedSQLServerAgentJobsCount
RDS データベースクラスター	すべて	CPUUtilization CommitLatency DatabaseConnections デッドロック FreeableMemory NetworkThroughput VolumeBytesUsed
Lambda 関数	すべて	所要時間 エラー IteratorAge ProvisionedConcurrencySpilloverInvocations スロットリング

コンポーネントタイプ	ワークロードタイプ	推奨されるメトリクス
SQS キュー	すべて	ApproximateAgeOfOldestMessage ApproximateNumberOfMessagesVisible NumberOfMessagesSent
Amazon DynamoDB テーブル	すべて	SystemErrors UserErrors ConsumedReadCapacityUnits ConsumedWriteCapacityUnits 調整イベントの読み込み 調整イベントの書き込み 条件チェックが失敗したリクエスト TransactionConflict

コンポーネントタイプ	ワークロードタイプ	推奨されるメトリクス
Amazon S3 バケット	すべて	レプリケーション時間制御 (RTC) を使用したレプリケーション設定が有効な場合: ReplicationLatency BytesPendingReplication OperationsPendingReplication リクエストメトリクスがオンになっている場合: 5xxErrors 4xxErrors BytesDownloaded BytesUploaded

コンポーネントタイプ	ワークロードタイプ	推奨されるメトリクス
AWS Step Functions	すべて	全般 • ExecutionThrottled • ExecutionsAborted • ProvisionedBucketSize • ProvisionedRefillRate • ConsumedCapacity 状態マシンタイプが **EXPRESS**、またはロググループレベルが **OFF** • ExecutionsFailed • ExecutionsTimedOut ステートマシンに Lambda 関数がある場合 • LambdaFunctionsFailed • LambdaFunctionsTimedOut ステートマシンにアクティビティがある場合 • ActivitiesFailed • ActivitiesTimedOut • ActivitiesHeartbeatTimedOut ステートマシンにサービス統合がある場合 • ServiceIntegrationsFailed

コンポーネントタイプ	ワークロードタイプ	推奨されるメトリクス
		• ServiceIntegration sTimedOut
API Gateway REST API ステージ	すべて	• 4XXErrors • 5XXErrors • レイテンシー

コンポーネントタイプ	ワークロードタイプ	推奨されるメトリクス
ECS クラスター	すべて	CpuUtilized MemoryUtilized NetworkRxBytes NetworkTxBytes RunningTaskCount PendingTaskCount StorageReadBytes StorageWriteBytes CPUReservation (EC2 起動タイプのみ) CPUUtilization (EC2 起動タイプのみ) MemoryReservation (EC2 起動タイプのみ) MemoryUtilization (EC2 起動タイプのみ) GPUReservation (EC2 起動タイプのみ) instance_cpu_utilization (EC2 起動タイプのみ) instance_filesystem_utilization (EC2 起動タイプのみ) instance_memory_utilization (EC2 起動タイプのみ)

コンポーネントタイプ	ワークロードタイプ	推奨されるメトリクス
		instance_network_total_bytes (EC2 起動タイプのみ)

コンポーネントタイプ	ワークロードタイプ	推奨されるメトリクス
	Java アプリケーション	CpuUtilized MemoryUtilized NetworkRxBytes NetworkTxBytes RunningTaskCount PendingTaskCount StorageReadBytes StorageWriteBytes CPUReservation (EC2 起動タイプのみ) CPUUtilization (EC2 起動タイプのみ) MemoryReservation (EC2 起動タイプのみ) MemoryUtilization (EC2 起動タイプのみ) GPUReservation (EC2 起動タイプのみ) instance_cpu_utilization (EC2 起動タイプのみ) instance_filesystem_utilization (EC2 起動タイプのみ) instance_memory_utilization (EC2 起動タイプのみ)

コンポーネントタイプ	ワークロードタイプ	推奨されるメトリクス
		instance_network_total_bytes (EC2 起動タイプのみ) java_lang_threading_threadcount java_lang_classloading_loadedclasscount java_lang_memory_heapmemoryusage_used java_lang_memory_heapmemoryusage_committed java_lang_operatingsystem_freephysicalmemorysize java_lang_operatingsystem_freeswapspacesize
ECS サービス	すべて	CPUUtilization MemoryUtilization CpuUtilized MemoryUtilized NetworkRxBytes NetworkTxBytes RunningTaskCount PendingTaskCount StorageReadBytes StorageWriteBytes

コンポーネントタイプ	ワークロードタイプ	推奨されるメトリクス
	Java アプリケーション	CPUUtilization MemoryUtilization CpuUtilized MemoryUtilized NetworkRxBytes NetworkTxBytes RunningTaskCount PendingTaskCount StorageReadBytes StorageWriteBytes java_lang_threading_threadcount java_lang_classloading_loadedclasscount java_lang_memory_heapmemoryusage_used java_lang_memory_heapmemoryusage_committed java_lang_operatingsystem_freephysicalmemorysize java_lang_operatingsystem_freeswapspacesize

コンポーネントタイプ	ワークロードタイプ	推奨されるメトリクス
EKS クラスター	すべて	cluster_failed_node_count
		node_cpu_reserved_capacity
		node_cpu_utilization
		node_filesystem_utilization
		node_memory_reserved_capacity
		node_memory_utilization
		node_network_total_bytes
		pod_cpu_reserved_capacity
		pod_cpu_utilization
		pod_cpu_utilization_over_pod_limit
		pod_memory_reserved_capacity
		pod_memory_utilization
		pod_memory_utilization_over_pod_limit
		pod_network_rx_bytes
		pod_network_tx_bytes

コンポーネントタイプ	ワークロードタイプ	推奨されるメトリクス
	Java アプリケーション	cluster_failed_node_count node_cpu_reserved_capacity node_cpu_utilization node_filesystem_utilization node_memory_reserved_capacity node_memory_utilization node_network_total_bytes pod_cpu_reserved_capacity pod_cpu_utilization pod_cpu_utilization_over_pod_limit pod_memory_reserved_capacity pod_memory_utilization pod_memory_utilization_over_pod_limit pod_network_rx_bytes pod_network_tx_bytes java_lang_threading_threadcount java_lang_classloading_loadedclasscount

コンポーネントタイプ	ワークロードタイプ	推奨されるメトリクス
		java_lang_memory_heapmemoryusage_used java_lang_memory_heapmemoryusage_committed java_lang_operatingsystem_freephysicalmemorysize java_lang_operatingsystem_freeswapspacesize

コンポーネントタイプ	ワークロードタイプ	推奨されるメトリクス
EC2 上の Kubernetes クラスター	すべて	cluster_failed_node_count node_cpu_reserved_capacity node_cpu_utilization node_filesystem_utilization node_memory_reserved_capacity node_memory_utilization node_network_total_bytes pod_cpu_reserved_capacity pod_cpu_utilization pod_cpu_utilization_over_pod_limit pod_memory_reserved_capacity pod_memory_utilization pod_memory_utilization_over_pod_limit pod_network_rx_bytes pod_network_tx_bytes

コンポーネントタイプ	ワークロードタイプ	推奨されるメトリクス
	Java アプリケーション	cluster_failed_node_count node_cpu_reserved_capacity node_cpu_utilization node_filesystem_utilization node_memory_reserved_capacity node_memory_utilization node_network_total_bytes pod_cpu_reserved_capacity pod_cpu_utilization pod_cpu_utilization_over_pod_limit pod_memory_reserved_capacity pod_memory_utilization pod_memory_utilization_over_pod_limit pod_network_rx_bytes pod_network_tx_bytes java_lang_threading_threadcount java_lang_classloading_loadedclasscount

コンポーネントタイプ	ワークロードタイプ	推奨されるメトリクス
		java_lang_memory_heapmemoryusage_used java_lang_memory_heapmemoryusage_committed java_lang_operatingsystem_freephysicalmemorysize java_lang_operatingsystem_freeswapspacesize

次の表は、コンポーネントタイプ別の推奨されるプロセスとプロセスメトリクスの一覧です。CloudWatch Application Insights は、インスタンスで実行されないプロセスのプロセスのモニターリングを推奨していません。

コンポーネントタイプ	ワークロードタイプ	推奨されるプロセス	推奨されるメトリクス

EC2 インスタンス (Windows サーバー)	マイクロソフト IIS/.NET Web フロントエンド	`w3wp`	`procstat cpu_usage ,`
			
			`procstat memory_rss ,`
			
			`procstat memory_vms ,`
			
			`procstat read_bytes ,`
			
			`procstat write_bytes`

	Microsoft SQL Server Database Tier	`SQLAgent`	`procstat cpu_usage ,`

コンポーネントタイプ	ワークロードタイプ	推奨されるプロセス	推奨されるメトリクス
			procstat memory_rss , procstat memory_vms , procstat read_bytes , procstat write_bytes
		sqlservr	procstat cpu_usage , procstat memory_rss , procstat memory_vms , procstat read_bytes , procstat write_bytes
		sqlwriter	procstat cpu_usage , procstat memory_rss

コンポーネントタイプ	ワークロードタイプ	推奨されるプロセス	推奨されるメトリクス
		Reporting ServicesService	procstat cpu_usage , procstat memory_rss
		MsDtsServr	procstat cpu_usage , procstat memory_rss , procstat memory_vms , procstat read_bytes , procstat write_bytes
		Msmdsrv	procstat cpu_usage , procstat memory_rss , procstat memory_vms , procstat read_bytes , procstat write_bytes

コンポーネントタイプ	ワークロードタイプ	推奨されるプロセス	推奨されるメトリクス
	.NET workerpool/Mid-Tier	w3wp	procstat cpu_usage , procstat memory_rss , procstat memory_vms , procstat read_bytes , procstat write_bytes
	.NET Core Tier	w3wp	procstat cpu_usage , procstat memory_rss , procstat memory_vms , procstat read_bytes , procstat write_bytes

パフォーマンスカウンタのメトリクス

パフォーマンスカウンタのメトリクスは、対応するパフォーマンスカウンタセットが Windows インスタンスにインストールされている場合にのみインスタンスに対して推奨されます。

パフォーマンスカウンタのメトリクス名	パフォーマンスカウンタセット名
.NET CLR Exceptions # of Exceps Thrown	.NET CLR Exceptions
.NET CLR Exceptions # of Exceps Thrown/Sec	.NET CLR Exceptions
.NET CLR Exceptions # of Filters/Sec	.NET CLR Exceptions
.NET CLR Exceptions # of Finallys/Sec	.NET CLR Exceptions
.NET CLR Exceptions Throw to Catch Depth/Sec	.NET CLR Exceptions
.NET CLR Interop # of CCWs	.NET CLR Interop
.NET CLR Interop # of Stubs	.NET CLR Interop
.NET CLR Interop # of TLB exports/Sec	.NET CLR Interop
.NET CLR Interop # of TLB imports/Sec	.NET CLR Interop
.NET CLR Interop # of Marshaling	.NET CLR Interop
.NET CLR Jit % Time in Jit	.NET CLR Jit
.NET CLR Jit Standard Jit Failures	.NET CLR Jit
.NET CLR Loading % Time Loading	.NET CLR Loading
.NET CLR Loading Rate of Load Failures	.NET CLR Loading
.NET CLR LocksAndThreads Contention Rate/Sec	.NET CLR LocksAndThreads
.NET CLR LocksAndThreads Queue Length/Sec	.NET CLR LocksAndThreads
.NET CLR Memory # Total Committed Bytes	.NET CLR Memory
.NET CLR Memory % Time in GC	.NET CLR Memory

パフォーマンスカウンタのメトリクス名	パフォーマンスカウンタセット名
.NET CLR Networking 4.0.0.0 HttpWebRequest Average Queue Time	.NET CLR Networking 4.0.0.0
.NET CLR Networking 4.0.0.0 HttpWebRequests Aborted/Sec	.NET CLR Networking 4.0.0.0
.NET CLR Networking 4.0.0.0 HttpWebRequests Failed/Sec	.NET CLR Networking 4.0.0.0
.NET CLR Networking 4.0.0.0 HttpWebRequests Queued/Sec	.NET CLR Networking 4.0.0.0
APP_POOL_WAS Total Worker Process Ping Failures	APP_POOL_WAS
ASP.NET Application Restarts	ASP.NET
ASP.NET Requests Rejected	ASP.NET
ASP.NET Worker Process Restarts	ASP.NET
ASP.NET Applications Cache API trims	ASP.NET Applications
ASP.NET Applications % Managed Processor Time (estimated)	ASP.NET Applications
ASP.NET Applications Errors Total/Sec	ASP.NET Applications
ASP.NET Applications Errors Unhandled During Execution/Sec	ASP.NET Applications
ASP.NET Applications Requests in Application Queue	ASP.NET Applications
ASP.NET Applications Requests/Sec	ASP.NET Applications
ASP.NET Request Wait Time	ASP.NET
ASP.NET Requests Queued	ASP.NET

パフォーマンスカウンタのメトリクス名	パフォーマンスカウンタセット名
Database ==> Instances Database Cache % Hit	Database ==> Instances
Database ==> Instances I/O Database Reads Average Latency	Database ==> Instances
Database ==> Instances I/O Database Reads/sec	Database ==> Instances
Database ==> Instances I/O Log Writes Average Latency	Database ==> Instances
DirectoryServices DRA Pending Replication Operations	DirectoryServices
DirectoryServices DRA Pending Replication Synchronizations	DirectoryServices
DirectoryServices LDAP Bind Time	DirectoryServices
DNS Recursive Queries/sec	DNS
DNS Recursive Query Failure/sec	DNS
DNS TCP Query Received/sec	DNS
DNS Total Query Received/sec	DNS
Total Response Sent/sec	DNS
DNS UDP Query Received/sec	DNS
HTTP Service Request Queues CurrentQueueSize	HTTP Service Request Queues
LogicalDisk % Free Space	LogicalDisk
LogicalDisk Avg。Disk sec/Write	LogicalDisk

パフォーマンスカウンタのメトリクス名	パフォーマンスカウンタセット名
LogicalDisk Avg。Disk sec/Read	LogicalDisk
LogicalDisk Avg。Disk Queue Length	LogicalDisk
Memory % Committed Bytes In Use	メモリ
Memory Available Mbytes	メモリ
Memory Pages/Sec	「メモリ」
Memory Long-Term Average Standby Cache Lifetime (s)	「メモリ」
Network Interface Bytes Total/sec	ネットワークインターフェイス
Network Interface Bytes Received/sec	ネットワークインターフェイス
Network Interface Bytes Sent/sec	ネットワークインターフェイス
Network Interface Current Bandwidth	ネットワークインターフェイス
Paging File % Usage	Paging File
PhysicalDisk % Disk Time	PhysicalDisk
PhysicalDisk Avg。Disk Queue Length	PhysicalDisk
PhysicalDisk Avg。Disk Sec/Read	PhysicalDisk
PhysicalDisk Avg。Disk Sec/Write	PhysicalDisk
PhysicalDisk Disk Read Bytes/Sec	PhysicalDisk
PhysicalDisk Disk Reads/Sec	PhysicalDisk
PhysicalDisk Disk Write Bytes/Sec	PhysicalDisk
PhysicalDisk Disk Writes/Sec	PhysicalDisk
Processor % Idle Time	プロセッサ

パフォーマンスカウンタのメトリクス名	パフォーマンスカウンタセット名
Processor % Interrupt Time	プロセッサ
Processor % Processor Time	プロセッサ
Processor % User Time	プロセッサ
SharePoint Disk-Based Cache Blob Cache fill ratio	SharePoint Disk-Based Cache
SharePoint Disk-Based Cache Blob cache flushes / second	SharePoint Disk-Based Cache
SharePoint Disk-Based Cache Blob cache hit ratio	SharePoint Disk-Based Cache
SharePoint Disk-Based Cache Total number of cache compactions	SharePoint Disk-Based Cache
SharePoint Foundation Executing Time/Page Request	SharePoint Foundation
SharePoint Publishing Cache Publishing cache flushes / second	SharePoint Publishing Cache
Security System-Wide Statistics Kerberos Authentications	セキュリティシステム全体の統計
Security System-Wide Statistics NTLM Authentications	セキュリティシステム全体の統計
SQLServer:Access Methods Forwarded Records/Sec	SQLServer:Access Methods
SQLServer:Access Methods Full Scans/Sec	SQLServer:Access Methods
SQLServer:Access Methods Page Splits/Sec	SQLServer:Access Methods

パフォーマンスカウンタのメトリクス名	パフォーマンスカウンタセット名
SQLServer:Buffer Manager Buffer cache hit Ratio	SQLServer:Buffer Manager
SQLServer:Buffer Manager Page life Expectancy	SQLServer:Buffer Manager
SQLServer:Database Replica File Bytes Received/sec	SQLServer:Database Replica
SQLServer:Database Replica Log Bytes Received/sec	SQLServer:Database Replica
SQLServer:Database Replica Log remaining for undo	SQLServer:Database Replica
SQLServer:Database Replica Log Send Queue	SQLServer:Database Replica
SQLServer:Database Replica Mirrored Write Transaction/sec	SQLServer:Database Replica
SQLServer:Database Replica Recovery Queue	SQLServer:Database Replica
SQLServer:Database Replica Redo Bytes Remaining	SQLServer:Database Replica
SQLServer:Database Replica Redone Bytes/ sec	SQLServer:Database Replica
SQLServer:Database Replica Total Log requiring undo	SQLServer:Database Replica
SQLServer:Database Replica Transaction Delay	SQLServer:Database Replica
SQLServer:General Statistics Processes Blocked	SQLServer:General Statistics

パフォーマンスカウンタのメトリクス名	パフォーマンスカウンタセット名
SQLServer:General Statistics User Connections	SQLServer:General Statistics
SQLServer:Latches Average Latch Wait Time (ms)	SQLServer:Latches
SQLServer:Locks Average Wait Time (ms)	SQLServer:Locks
SQLServer:Locks Lock Timeouts/Sec	SQLServer:Locks
SQLServer:Locks Lock Waits/Sec	SQLServer:Locks
SQLServer:Locks Number of Deadlocks/Sec	SQLServer:Locks
SQLServer:Memory Manager Memory Grants Pending	SQLServer:Memory Manager
SQLServer:SQL Statistics Batch Requests/Sec	SQLServer:SQL Statistics
SQLServer:SQL Statistics SQL Compilations/Sec	SQLServer:SQL Statistics
SQLServer:SQL Statistics SQL Re-Compilations/Sec	SQLServer:SQL Statistics
System Processor Queue Length	システム
TCPv4 Connections Established	TCPv4
TCPv6 Connections Established	TCPv6
W3SVC_W3WP File Cache Flushes	W3SVC_W3WP
W3SVC_W3WP File Cache Misses	W3SVC_W3WP
W3SVC_W3WP Requests/Sec	W3SVC_W3WP
W3SVC_W3WP URI Cache Flushes	W3SVC_W3WP

パフォーマンスカウンタのメトリクス名	パフォーマンスカウンタセット名
W3SVC_W3WP URI Cache Misses	W3SVC_W3WP
Web Service Bytes Received/Sec	ウェブサービス
Web Service Bytes Sent/Sec	ウェブサービス
Web Service Connection Attempts/Sec	ウェブサービス
Web Service Current Connections	ウェブサービス
Web Service Get Requests/Sec	ウェブサービス
Web Service Post Requests/Sec	ウェブサービス

CloudWatch コンソールのリソースのヘルスビューの使用

リソースのヘルスビューを使用すると、アプリケーション全体のホストのヘルスとパフォーマンスを単一のビューで自動的に検出、管理、および可視化できます。CPU やメモリなどのパフォーマンスディメンションでホストのヘルスを可視化し、フィルターを使用して数百のホストを単一のビューで詳しく分析できます。タグやユースケース (同じ Auto Scaling グループ内のホスト、または同じロードバランサーを使用するホストなど) でフィルタリングできます。

前提条件

リソースのヘルスビューの利点を最大限に活用するには、次の前提条件を満たしていることを確認します。

- ホストのメモリ使用率を確認し、フィルターとして使用するには、CloudWatch エージェントをホストにインストールし、デフォルトの `CWAgent` 名前空間で CloudWatch にメモリメトリクスを送信するように設定する必要があります。Linux および macOS インスタンスでは、CloudWatch エージェントが `mem_used_percent` メトリクスを送信する必要があります。Windows インスタンスでは、エージェントは `Memory % Committed Bytes In Use` メトリクスを送信する必要があります。これらのメトリクスは、ウィザードを使用して CloudWatch エージェント設定ファイルを作成し、事前定義されたメトリクスのセットのいずれかを選択した場合に含まれます。CloudWatch エージェントによって収集されたメトリクスは、カスタムメトリクスとして請求されます。詳細については、「CloudWatch エージェントのインストール」を参照してください。

CloudWatch エージェントを使用してこれらのメモリメトリクスを収集し、リソースヘルスビューで使用する場合は、CloudWatch エージェント設定ファイルに次のセクションを含める必要があります。このセクションにはデフォルトのディメンション設定が含まれており、デフォルト値で作成されるため、このセクションのどの部分も、次の例に示されているのと異なるものに変更しないでください。

```
"append_dimensions": {
  "ImageId": "${aws:ImageId}",
  "InstanceId": "${aws:InstanceId}",
  "InstanceType": "${aws:InstanceType}",
  "AutoScalingGroupName": "${aws:AutoScalingGroupName}"
},
```

- リソースのヘルスビューで使用可能なすべての情報を表示するには、次の許可を持つアカウントにサインインする必要があります。より少ない許可でサインオンしている場合でも、リソースのヘルスビューを使用できますが、一部のパフォーマンスデータは表示されません。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "autoscaling:Describe*",
                "cloudwatch:Describe*",
                "cloudwatch:Describe*",
                "cloudwatch:Get*",
                "cloudwatch:List*",
                "logs:Get*",
                "logs:Describe*",
                "sns:Get*",
                "sns:List*",
                "ec2:DescribeInstances",
                "ec2:DescribeInstanceStatus",
                "ec2:DescribeRegions"
            ],
            "Effect": "Allow",
            "Resource": "*"
        }
    ]
}
```

アカウントのリソースのヘルスを表示するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで、[アプリケーションのモニタリング]、[リソースのヘルス] の順に選択します。

   リソースのヘルスページが表示され、アカウント内のホストごとに四角形が表示されます。各四角形は、[Color by] (色分け) の設定に基づいて、そのホストの現在のステータスに基づいて色分けされます。アラーム記号が付されたホストの四角形には、現在 ALARM 状態にある 1 つ以上のアラームがあります。

   1 つのビューで最大 500 のホストを表示できます。アカウントでさらにホストを有している場合は、この手順のステップ 6 のフィルター設定を使用します。
3. 各ホストのヘルスの表示に使用する条件を変更するには、[Color by] (色分け) の設定を選択します。[CPU Utilization] (CPU 使用率)、[Memory Utilization] (メモリ使用率)、または [Status check] (ステータスチェック) を選択できます。メモリ使用率のメトリクスは、CloudWatch エージェントを実行しており、メモリメトリクスを収集してデフォルトの `CWAgent` 名前空間に送信するように設定されているホストでのみ使用できます。詳細については、「CloudWatch エージェントを使用してメトリクス、ログ、トレースを収集する」を参照してください。
4. グリッド内のヘルスインジケータに使用されるしきい値および色を変更するには、グリッドの上にある歯車アイコンを選択します。
5. ホストグリッドにアラームを表示するかどうかを切り替えるには、[Show alarms across all metrics] (すべてのメトリクスでアラームを表示) をオンまたはオフにします。
6. マップ内のホストをグループに分割するには、[Group by] (グループ化の条件) でグループ化の条件を選択します。
7. ビューで表示されるホストを絞り込むには、[Filter by] (フィルター条件) でフィルター条件を選択します。タグや Auto Scaling グループ、インスタンスタイプ、セキュリティグループなどのリソースグループでフィルタリングできます。
8. ホストを並べ替えるには、[Sort by] (並べ替え) で並べ替え条件を選択します。ステータスチェックの結果、インスタンスの状態、CPU またはメモリ使用率、ALARM 状態にあるアラームの数でソートできます。
9. ホストの詳細を表示するには、そのホストを表す四角形を選択します。ポップアップペインが表示されます。その後、そのホストに関する情報をより詳細に確認するには、[View dashboard] (ダッシュボードを表示) または [View on list] (リストで表示) を選択します。

CloudWatch のクロスアカウントオブザーバビリティ

Amazon CloudWatch のクロスアカウントオブザーバビリティを使用すると、リージョン内の複数のアカウントにまたがるアプリケーションをモニターリングおよびトラブルシューティングできます。アカウントの境界をなくすことで、リンクされたすべてのアカウントのメトリクス、ログ、トレース、Application Insights アプリケーション、Internet Monitor のモニターをシームレスに検索、可視化、分析できます。

1 つ以上の AWS アカウントをモニターリングアカウントとして設定し、それらを複数のソースアカウントにリンクします。モニターリングアカウントは、ソースアカウントから生成されたオブザーバビリティデータを表示して操作できる中心的な AWS アカウントです。ソースアカウントは、そのアカウント内にあり、リソースのオブザーバビリティデータを生成する個々の AWS アカウントです。ソースアカウントは、オブザーバビリティデータをモニターリングアカウントと共有します。共有されるオブザーバビリティデータには、次のタイプのテレメトリが含まれます。

- Amazon CloudWatch のメトリクス。すべての名前空間のメトリクスをモニタリングアカウントと共有できるほか、フィルターで一部の名前空間に絞り込むことができます。
- Amazon CloudWatch Logs のロググループ。すべてのロググループをモニタリングアカウントと共有できるほか、フィルターで一部のロググループに絞り込むことができます。
- AWS X-Ray のトレース
- Amazon CloudWatch Application Insights のアプリケーション
- CloudWatch Internet Monitor でのモニタリング

モニターリングアカウントとソースアカウント間のリンクを作成するには、CloudWatch コンソールを使用できます。または、AWS CLI および API の Observability Access Manager コマンドを使用します。詳細については、「Observability Access Manager API Reference」(Observability Access Manager API リファレンス) を参照してください。

シンクは、モニターリングアカウントのアタッチメントポイントを表すリソースです。ソースアカウントは、シンクにリンクすることでオブザーバビリティデータを共有できます。各アカウントにリージョンあたり 1 つのシンクを設定できます。各シンクは、そのシンクがあるモニターリングアカウントによって管理されます。オブザーバビリティリンクは、ソースアカウントとモニターリングアカウントの間に確立されたリンクを表すリソースです。リンクはソースアカウントによって管理されます。

CloudWatch のクロスアカウントオブザーバビリティを設定するデモについては、次の動画を参照してください。

次のトピックでは、モニターリングアカウントとソースアカウントの両方で、CloudWatch のクロスアカウントオブザーバビリティを設定する方法について説明します。クロスアカウントクロスリージョンの CloudWatch ダッシュボードについては、「クロスアカウントクロスリージョン CloudWatch コンソール」を参照してください。

ソースアカウントに Organizations を使用する

ソースアカウントをモニターリングアカウントにリンクする方法は 2 つあります。そのうちの 1 つまたは両方を使用できます。

- AWS Organizations を使用して、1 つの組織または組織単位内のアカウントをモニターリングアカウントにリンクします。
- 個々の AWS アカウントをモニターリングアカウントに接続します。

組織で後から作成される新しい AWS アカウントが、ソースアカウントとしてクロスアカウントオブザーバビリティに自動的にオンボーディングされるよう、Organizations を使用することをお勧めします。

モニターリングアカウントとソースアカウントのリンクに関する詳細

- 各モニターリングアカウントは、最大 100,000 個のソースアカウントにリンクできます。
- 各ソースアカウントは、最大 5 個のモニターリングアカウントとデータを共有できます。
- 1 つのアカウントを、モニターリングアカウントとソースアカウントの両方に設定できます。その場合、このアカウント自体からのオブザーバビリティデータのみがリンクされたモニターリングアカウントに送信されます。他のソースアカウントからのデータは中継されません。
- モニターリングアカウントは、共有可能なテレメトリのタイプを指定します。ソースアカウントは、共有するテレメトリのタイプを指定します。
  - モニターリングアカウントで選択されているテレメトリのタイプがソースアカウントよりも多い場合、アカウントはリンクされます。両方のアカウントで選択されたデータのタイプのみが共有されます。
  - ソースアカウントで選択されているテレメトリのタイプがモニターリングアカウントよりも多い場合、リンクの作成は失敗し、何も共有されません。

- メトリクス名は、リンクの作成後に該当するメトリクスが新しいデータポイントを出力するまで、モニタリングアカウントコンソールに表示されません。
- アカウント間のリンクの削除は、ソースアカウントから行います。
- モニタリングアカウントのシンクを削除するには、まずモニタリングアカウントのそのシンクへのリンクをすべて削除する必要があります。

料金表

CloudWatch のクロスアカウントオブザーバビリティでは、ログとメトリクスの追加コストは発生せず、最初のトレースコピーも無料です。料金の詳細については、「Amazon CloudWatch の料金」を参照してください。

目次

モニターリングアカウントをソースアカウントにリンクする

このセクションのトピックでは、モニターリングアカウントとソースアカウント間のリンクを設定する方法について説明します。

組織のモニターリングアカウントとして機能する新しい AWS アカウントを作成することをお勧めします。

目次

必要なアクセス許可

モニターリングアカウントとソースアカウント間のリンクを作成するには、特定のアクセス許可を使用してサインインする必要があります。

- モニターリングアカウントを設定するには — モニターリングアカウントの完全な管理者アクセス権を持っているか、次のアクセス許可を使用してそのアカウントにサインインする必要があります。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowSinkModification",
            "Effect": "Allow",
            "Action": [
                "oam:CreateSink",
                "oam:DeleteSink",
                "oam:PutSinkPolicy",
                "oam:TagResource"
            ],
            "Resource": "*"
        },
        {
```

```
            "Sid": "AllowReadOnly",
            "Effect": "Allow",
            "Action": ["oam:Get*", "oam:List*"],
            "Resource": "*"
        }
    ]
}
```

- 特定のモニターリングアカウントにスコープされるソースアカウント — 指定した 1 つのモニターリングアカウントのみのリンクを作成、更新、管理するには、少なくとも次のアクセス許可を持つアカウントにサインインする必要があります。この例では、モニターリングアカウントは 999999999999 です。

  リンクが 5 つのリソースタイプ (メトリクス、ログ、トレース、Application Insights アプリケーション、Internet Monitor モニター) のいずれかを共有していない場合は、必要に応じて `cloudwatch:Link`、`logs:Link`、`xray:Link`、`applicationinsights:Link`、または `internetmonitor:Link` を省略できます。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "oam:CreateLink",
                "oam:UpdateLink",
                "oam:DeleteLink",
                "oam:GetLink",
                "oam:TagResource"
            ],
            "Effect": "Allow",
            "Resource": "arn:*:oam:*:*:link/*"
        },
        {
            "Action": [
                "oam:CreateLink",
                "oam:UpdateLink"
            ],
            "Effect": "Allow",
            "Resource": "arn:*:oam:*:*:sink/*",
            "Condition": {
                "StringEquals": {
                    "aws:ResourceAccount": [
```

```
                        "999999999999"
                    ]
                }
            }
        },
        {
            "Action": "oam:ListLinks",
            "Effect": "Allow",
            "Resource": "*"
        },
        {
            "Action": "cloudwatch:Link",
            "Effect": "Allow",
            "Resource": "*"
        },
        {
            "Action": "logs:Link",
            "Effect": "Allow",
            "Resource": "*"
        },
        {
            "Action": "xray:Link",
            "Effect": "Allow",
            "Resource": "*"
        },
        {
             "Action": "applicationinsights:Link",
             "Effect": "Allow",
             "Resource": "*"
         },
        {
             "Action": "internetmonitor:Link",
             "Effect": "Allow",
             "Resource": "*"
        }
    ]
}
```

- 任意のモニタリングアカウントにリンクできるアクセス許可を持つソースアカウント — 既存のモニタリングアカウントのシンクへのリンクを作成し、メトリクス、ロググループ、トレース、Application Insights アプリケーション、Internet Monitor モニターを共有するには、完全な管理者アクセス許可または次のアクセス許可で、ソースアカウントにサインインする必要があります。

リンクが 5 つのリソースタイプ (メトリクス、ログ、トレース、Application Insights アプリケーション、Internet Monitor モニター) のいずれかを共有していない場合は、必要に応じて `cloudwatch:Link`、`logs:Link`、`xray:Link`、`applicationinsights:Link`、または `internetmonitor:Link` を省略できます。

```
{
    "Version": "2012-10-17",
    "Statement": [{
            "Effect": "Allow",
            "Action": [
                "oam:CreateLink",
                "oam:UpdateLink"
            ],
            "Resource": [
                "arn:aws:oam:*:*:link/*",
                "arn:aws:oam:*:*:sink/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "oam:List*",
                "oam:Get*"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "oam:DeleteLink",
                "oam:GetLink",
                "oam:TagResource"
            ],
            "Resource": "arn:aws:oam:*:*:link/*"
        },
        {
            "Action": "cloudwatch:Link",
            "Effect": "Allow",
            "Resource": "*"
        },
        {
            "Action": "xray:Link",
```

```
            "Effect": "Allow",
            "Resource": "*"
        },
        {
            "Action": "logs:Link",
            "Effect": "Allow",
            "Resource": "*"
        },
        {
             "Action": "applicationinsights:Link",
             "Effect": "Allow",
             "Resource": "*"
        },
        {
             "Action": "internetmonitor:Link",
             "Effect": "Allow",
             "Resource": "*"
        }
    ]
}
```

設定の概要

次の大まかな手順では、CloudWatch のクロスアカウントオブザーバビリティを設定する方法を示します。

> **Note**
>
> 組織のモニターリングアカウントとして使用する新しい AWS アカウントを作成することをお勧めします。

1. 専用のモニターリングアカウントを設定します。
2. (オプション) AWS CloudFormation テンプレートをダウンロードするか URL をコピーして、ソースアカウントをリンクします。
3. ソースアカウントをモニターリングアカウントにリンクします。

これらの手順を完了すると、モニターリングアカウントを使用してソースアカウントのオブザーバビリティデータを表示できます。

ステップ1: モニターリングアカウントを設定する

このセクションの手順に従って、CloudWatch のクロスアカウントオブザーバビリティ用のモニターリングアカウントとして AWS アカウントを設定します。

前提条件

- ソースアカウントとして AWS Organizations 組織内のアカウントを設定する場合 — 組織パスまたは組織 ID を取得します。
- ソースアカウントに Organizations を使用していない場合 — ソースアカウントのアカウント ID を取得します。

アカウントをモニターリングアカウントとして設定するには、特定のアクセス許可が必要です。詳細については、「必要なアクセス許可」を参照してください。

モニターリングアカウントを設定するには

1. モニターリングアカウントとして使用するアカウントにサインインします。
2. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
3. 左側のナビゲーションペインで [設定] を選択します。
4. [Monitoring account configuration] (モニターリングアカウント設定) で、[Configure] (設定) を選択します。
5. [データの選択] で、このモニタリングアカウントが、リンクされているソースアカウントからの [ログ]、[メトリクス]、[トレース]、[Application Insights - Applications]、[Internet Monitor - モニター] のデータを表示できるようにするかどうかを選択します。
6. [List source accounts] (ソースアカウントを一覧表示) に、このモニターリングアカウントが表示するソースアカウントを入力します。ソースアカウントを特定するには、個々のアカウント ID、組織パス、または組織 ID を入力します。組織パスまたは組織 ID を入力すると、その組織内のすべてのリンクされたアカウントからのオブザーバビリティデータをモニターリングアカウントで表示できます。

   リストのエントリはカンマで区切ります。

> **Important**
>
> 組織パスを入力するときは、正確な形式に従います。ou-id は / (スラッシュ文字) で終わる必要があります。例: `o-a1b2c3d4e5/r-f6g7h8i9j0example/ou-def0-awsbbbb/`

7. [Define a label to identify your source account] (ラベルを定義してソースアカウント特定する) で、モニターリングアカウントを使用してソースアカウントを表示する際に、アカウント名またはメールアドレスのどちらを使用してソースアカウントを特定するかを指定します。
8. [設定] を選択します。

> **Important**
>
> モニターリングアカウントとソースアカウント間のリンクは、ソースアカウントを設定するまで完了しません。詳細については、次のセクションを参照してください。

ステップ 2: (オプション) AWS CloudFormation テンプレートまたは URL をダウンロードする

ソースアカウントをモニターリングアカウントにリンクするには、AWS CloudFormation テンプレートまたは URL を使用することをお勧めします。

- 組織全体をリンクする場合 — CloudWatch では、AWS CloudFormation テンプレートが提供されています。
- 個々のアカウントをリンクする場合 — CloudWatch が提供する URL または AWS CloudFormation テンプレートを使用します。

AWS CloudFormation テンプレートを使用するには、次の手順でテンプレートをダウンロードする必要があります。モニターリングアカウントを少なくとも 1 つのソースアカウントにリンクすると、AWS CloudFormation テンプレートをダウンロードできなくなります。

AWS CloudFormation テンプレートをダウンロードまたは URL をコピーして、ソースアカウントをモニターリングアカウントにリンクするには

1. モニターリングアカウントとして使用するアカウントにサインインします。

2. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
3. 左側のナビゲーションペインで [設定] を選択します。
4. [Monitoring account configuration] (モニターリングアカウント設定) から、[Resources to link accounts] (アカウントをリンクするためのリソース) を選択します。
5. 次のいずれかを行います。
   - [AWS 組織] を選択してテンプレートを入手し、組織内のアカウントをこのモニタリングアカウントにリンクするために使用します。
   - [Any account] (任意のアカウント) を選択して、個々のアカウントをソースアカウントとして設定するためのテンプレートまたは URL を入手します。
6. 次のいずれかを行います。
   - [AWS 組織] を選択した場合は、[CloudFormation テンプレートをダウンロード] を選択します。
   - [Any account] (任意のアカウント) を選択した場合は、[Download CloudFormation template] (CloudFormation テンプレートのダウンロード) または [Copy URL] (URL のコピー) を選択します。
7. (オプション) 5 から 6 の手順を繰り返し、AWS CloudFormation テンプレートと URL の両方をダウンロードします。

ステップ 3: ソースアカウントをリンクする

次のセクションの手順を使用して、ソースアカウントをモニターリングアカウントにリンクします。

モニターリングアカウントをソースアカウントにリンクするには、特定のアクセス許可が必要です。詳細については、「必要なアクセス許可」を参照してください。

AWS CloudFormation テンプレートを使用して、組織または組織単位内のすべてのアカウントをソースアカウントとして設定する

これらの手順は、ステップ 2: (オプション) AWS CloudFormation テンプレートまたは URL をダウンロードする の手順を実行し、必要な AWS CloudFormation テンプレートを既にダウンロードしていることを前提としています。

AWS CloudFormation テンプレートを使用して、組織または組織単位内のアカウントをモニタリングアカウントにリンクするには

1. 組織のモニタリングアカウントにサインインします。
2. https://console.aws.amazon.com/cloudformation で AWS CloudFormation コンソール を開きます。
3. 左のナビゲーションバーから [StackSets] を選択します。
4. 目的のリージョンにサインインしていることを確認し、[Create StackSet] (StackSet の作成) を選択します。
5. [Next] を選択します。
6. [Template is ready] (テンプレートの準備完了) を選択し、[Upload a template file] (テンプレートファイルをアップロード) を選択します。
7. [Choose file] (ファイルを選択) を選択し、モニタリングアカウントからダウンロードしたテンプレートを選択して、[Open] (開く) をクリックします。
8. [Next] を選択します。
9. [Specify StackSet details] (StackSet の詳細を指定) で、StackSet の名前を入力して [Next] (次へ) をクリックします。
10. [Add stacks to stack set] (スタックセットにスタックを追加) で、[Deploy new stacks] (新しいスタックのデプロイ) を選択します。
11. [Deployment targets] (デプロイターゲット) で、組織全体にデプロイするか、特定の組織単位にデプロイするかを選択します。
12. [Specify regions] (リージョンを指定) で、CloudWatch のクロスアカウントオブザーバビリティをデプロイするリージョンを選択します。
13. [Next] を選択します。
14. [Review] (確認) ページで、選択内容を確認し [Submit] (送信) をクリックします。
15. [Stack instances] (スタックインスタンス) タブで、スタックインスタンスのステータスが CREATE_COMPLETE になるまで画面を更新します。

AWS CloudFormation テンプレートを使用して個別のソースアカウントを設定する

これらの手順は、ステップ 2: (オプション) AWS CloudFormation テンプレートまたは URL をダウンロードする の手順を実行し、必要な AWS CloudFormation テンプレートを既にダウンロードしていることを前提としています。

AWS CloudFormation テンプレートを使用して、CloudWatch のクロスアカウントオブザーバビリティ用に個別のソースアカウントを設定するには

1. ソースアカウントにサインインします。
2. https://console.aws.amazon.com/cloudformation で AWS CloudFormation コンソール を開きます。
3. 左のナビゲーションバーで [Stacks] (スタック) を選択します。
4. 目的のリージョンにログインしていることを確認し、[Create stack] (スタックの作成)、[With new resources (standard)] (新しいリソースを使用 (標準)) を選択します。
5. [Next] を選択します。
6. [Upload a template file (テンプレートファイルをアップロード)] を選択します。
7. [Choose file] (ファイルを選択) を選択し、モニターリングアカウントからダウンロードしたテンプレートを選択して、[Open] (開く) をクリックします。
8. [Next] を選択します。
9. [Specify stack details] (スタックの詳細を指定) ページで、スタックの名前を入力して [Next] (次へ) をクリックします。
10. [スタックオプションの設定] ページで、[次へ] をクリックします。
11. [Review] (確認) ページで [Submit] (送信) をクリックします。
12. スタックのステータスのページで、ステータスが CREATE_COMPLETE になるまで画面を更新します。
13. 同じテンプレートを使用して他のソースアカウントをこのモニターリングアカウントにリンクするには、このアカウントからサインアウトし、次のソースアカウントにサインインします。その後、2 から 12 の手順を繰り返します。

URL を使用して個別のソースアカウントを設定する

これらの手順は、ステップ 2: (オプション) AWS CloudFormation テンプレートまたは URL をダウンロードする の手順を実行し、必要な URL を既にコピーしていることを前提としています。

URL を使用して個々のソースアカウントをモニターリングアカウントにリンクするには

1. ソースアカウントとして使用するアカウントにサインインします。
2. モニターリングアカウントからコピーした URL を入力します。

   いくつかの情報が入力された CloudWatch の設定ページが表示されます。

3. [データの選択] で、このソースアカウントが [ログ]、[メトリクス]、[トレース]、[Application Insights - Applications]、[Internet Monitor - モニター] のデータをモニタリングアカウントと共有するかどうかを選択します。

   [ログ] と [メトリクス] では、リソースの全部または一部をモニタリングアカウントと共有するかどうかを選択できます。

   a. (オプション) このアカウントのロググループのサブセットをモニタリングアカウントと共有するには、[ログ] を選択し、[ログのフィルタリング] を選択します。次に、[ログのフィルタリング] ボックスを使用して、共有するロググループを検索するクエリを作成します。そのクエリでは、条件 LogGroupName と次の 1 つ以上のオペランドを使用します。

      - = および !=
      - AND
      - OR
      - ^ は LIKE を示し、!^ は NOT LIKE を示します。どちらも、プレフィックス検索としてのみ使用できます。検索して見つかった文字列を含める場合は、その文字列の末尾に % を含めます。
      - IN と NOT IN。括弧 (( )) を使用します。

      クエリ全体の文字数は 2,000 文字以内にする必要があり、指定できる条件オペランドは 5 つまでに制限されています。条件オペランドには、AND と OR があります。他のオペランドの数に上限はありません。

      > **Tip**
      >
      > [サンプルクエリを表示] を選択すると、よく使用されるクエリ形式に適した構文が表示されます。

   b. (オプション) このアカウントのメトリクス名前空間のサブセットをモニタリングアカウントと共有するには、[メトリクス] を選択し、[メトリクスのフィルタリング] を選択します。次に、[メトリクスのフィルタリング] ボックスを使用して、共有するメトリクス名前空間を検索するクエリを作成します。条件 Namespace と次の 1 つ以上のオペランドを使用します。

      - = および !=
      - AND

- OR
- LIKE および NOT LIKE。どちらも、プレフィックス検索としてのみ使用できます。検索して見つかった文字列を含める場合は、その文字列の末尾に % を含めます。
- IN と NOT IN 。括弧 (( )) を使用します。

クエリ全体の文字数は 2,000 文字以内にする必要があり、指定できる条件オペランドは 5 つまでに制限されています。条件オペランドには、AND と OR があります。他のオペランドの数に上限はありません。

> **Tip**
>
> [サンプルクエリを表示] を選択すると、よく使用されるクエリ形式に適した構文が表示されます。

4. [Enter monitoring account configuration ARN] (モニターリングアカウント設定 ARN を入力) で ARN を変更しないでください。
5. [Define a label to identify your source account] (ラベルを定義してソースアカウントを特定する) セクションには、モニターリングアカウントで選択したラベルがあらかじめ入力されています。必要に応じて、[Edit] (編集) を選択し変更します。
6. [Link (リンク)] を選択します。
7. ボックスに「**Confirm**」と入力し、[Confirm] (確認) をクリックします。
8. 同じ URL を使用して他のソースアカウントをこのモニターリングアカウントにリンクするには、このアカウントからサインアウトし、次のソースアカウントにログインします。その後、2 から 7 の手順を繰り返します。

モニターリングアカウントとソースアカウントを管理する

モニターリングアカウントとソースアカウントを設定したら、次のセクションの手順を実行してそれらを管理できます。

目次

モニターリングアカウントにソースアカウントを追加でリンクする

このセクションの手順を実行して、他のソースアカウントから既存のモニターリングアカウントにリンクを追加できます。

各ソースアカウントは、最大 5 つのモニターリングアカウントにリンクできます。各モニターリングアカウントは、最大 100,000 個のソースアカウントにリンクできます。

ソースアカウントを管理するには、特定のアクセス許可が必要です。詳細については、「必要なアクセス許可」を参照してください。

モニターリングアカウントにソースアカウントをさらに追加するには

1. モニターリングアカウントにサインインします。
2. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
3. 左側のナビゲーションペインで [設定] を選択します。
4. [Monitoring account configuration] (モニターリングアカウント設定) から、[Manage source accounts] (ソースアカウントを管理) を選択します。
5. [Configuration policy] (設定ポリシー) タブを選択します。
6. [Configuration policy] (設定ポリシー) ボックスの [Principal] (プリンシパル) 行に新しいソースアカウント ID を追加します。

   例えば、現在 [Principal] (プリンシパル) 行が次のようになっているとします。

   ```
   "Principal": {"AWS": ["111111111111", "222222222222"]}
   ```

   3 番目のソースアカウントとして 999999999999 を追加するには、次のように行を編集します。

   ```
   "Principal": {"AWS": ["111111111111", "222222222222", "999999999999"]}
   ```
7. [Update] (更新) を選択します。
8. [Configuration set] (設定の詳細) タブを選択します。
9. モニターリングアカウントのシンクの ARN の横にあるコピーアイコンを選択します。
10. 新しいソースアカウントとして使用するアカウントにサインインします。
11. ステップ 9 でコピーしたモニタリングアカウントのシンク ARN を貼り付けます。

いくつかの情報が入力された CloudWatch の設定ページが表示されます。

12. [データの選択] で、このソースアカウントが [ログ]、[メトリクス]、[トレース]、[Application Insights - Applications] のデータをリンク先のモニタリングアカウントに送信するかどうかを選択します。
13. [Enter monitoring account configuration ARN] (モニターリングアカウント設定 ARN を入力) で ARN を変更しないでください。
14. [Define a label to identify your source account] (ラベルを定義してソースアカウントを特定する) セクションには、モニターリングアカウントで選択したラベルがあらかじめ入力されています。必要に応じて、[Edit] (編集) を選択し変更します。
15. [Link (リンク)] を選択します。
16. ボックスに「**`Confirm`**」と入力し、[Confirm] (確認) をクリックします。

モニターリングアカウントとソースアカウント間のリンクを削除する

ソースアカウントからモニターリングアカウントへのデータの送信を停止するには、このセクションの手順を実行します。

このタスクを完了するには、ソースアカウントの管理に必要となるアクセス許可が必要です。詳細については、「必要なアクセス許可」を参照してください。

ソースアカウントとモニターリングアカウント間のリンクを削除するには

1. ソースアカウントにサインインします。
2. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
3. 左側のナビゲーションペインで [設定] を選択します。
4. [Source account information] (ソースアカウント情報) で、[View monitoring accounts] (モニターリングアカウントを表示) を選択します。
5. データの共有を停止するモニターリングアカウントの横にあるチェックボックスをオンします。
6. [Stop sharing data] (データの共有を停止する)、[Confirm] (確認) の順を選択します。
7. モニターリングアカウントにサインインします。
8. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
9. [設定] を選択します。
10. [Monitoring account information] (モニターリングアカウント情報) で、[View configuration] (設定の表示) を選択します。

11. [Policy] (ポリシー) ボックスで、[Principal] (プリンシパル) 行からソースアカウント ID を削除し、[Update] (更新) をクリックします。

モニターリングアカウントに関する情報を表示する

モニターリングアカウントのクロスアカウント設定を表示するには、このセクションの手順を実行します。

モニターリングアカウントを管理するには、特定のアクセス許可が必要です。詳細については、「必要なアクセス許可」を参照してください。

モニターリングアカウントを管理するには

1. モニターリングアカウントにサインインします。
2. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
3. 左側のナビゲーションペインで [設定] を選択します。
4. [Monitoring account configuration] (モニターリングアカウント設定) から、[Manage source accounts] (ソースアカウントを管理) を選択します。
5. このアカウントをモニターリングアカウントとして有効にする Observability Access Manager ポリシーを表示するには、[Configuration policy] (設定ポリシー) タブを選択します。
6. このモニターリングアカウントにリンクされているソースアカウントを表示するには、[Linked source accounts] (リンクされたソースアカウント) タブを選択します。
7. モニターリングアカウントのシンクの ARN、およびモニターリングアカウントがリンクされているソースアカウントで表示できるデータのタイプを確認するには、[Linked source accounts] (リンクされたソースアカウント) タブを選択します。

他のデータソースにあるメトリクスへのクエリ

CloudWatch では、他のデータソースにあるメトリクスのアラームをクエリ、可視化、作成できます。そのためには、CloudWatch を他のデータソースに接続します。これにより、CloudWatch コンソール内に単一の統合モニタリングエクスペリエンスを実現できます。データの保存場所に関係なく、インフラストラクチャとアプリケーションのメトリクスを一元的に表示できるため、問題をすばやく特定して解決できます。

CloudWatch ウィザードを使用してデータソースに接続すると、AWS CloudFormation スタックが作成されるので、AWS Lambda 関数をデプロイして設定できるようになります。この Lambda 関数は、データソースをクエリするたびにオンデマンドで実行されます。CloudWatch クエリビルダーには、メトリクス、テーブル、フィールド、ラベルなど、クエリ可能な要素のリストがリアルタイムに表示されます。何かを選択すると、選択したソースにネイティブの言語でクエリが事前に入力されます。

CloudWatch では、ウィザードの手順に従って、以下のデータソースに接続できます。データソースと認証情報を特定できるように、各データソースの基本情報を指定できます。また、独自の Lambda 関数を作成して、他のデータソースへのコネクタを手動で作成することもできます。

- Amazon OpenSearch Service - OpenSearch Service のログとトレースからメトリクスを導出します。
- Amazon Managed Service for Prometheus - PromQL を使用して、こうしたメトリクスをクエリします。
- Amazon RDS for MySQL - SQL を使用して、Amazon RDS テーブルに保存されているデータをメトリクスに変換します。
- Amazon RDS for PostgreSQL - SQL を使用して、Amazon RDS テーブルに保存されているデータをメトリクスに変換します。
- Amazon S3 CSV ファイル - Amazon S3 バケットに保存されている CSV ファイルのメトリクスデータを表示します。
- Microsoft Azure Monitor - Microsoft Azure Monitor アカウントからメトリクスをクエリします。
- Prometheus - PromQL を使用して、こうしたメトリクスをクエリします。

データソースへのコネクタを作成したら、「別のデータソースにあるメトリクスのグラフ化」でデータソースのメトリクスをグラフ化する方法を確認してください。データソースのメトリクスにアラームを設定する方法については、「接続されたデータソースに基づいてアラームを作成する」を参照してください。

トピック

データソースへのアクセスの管理

CloudWatch は、AWS CloudFormation を使用して、アカウントで必要になるリソースを作成します。IAM ユーザーに `CreateStack` アクセス許可を付与する場合は、`cloudformation:TemplateUrl` 条件を使用して AWS CloudFormation テンプレートへのアクセスを制御することをお勧めします。

> **Warning**
>
> データソース呼び出しアクセス許可をユーザーに付与した場合、そのユーザーはデータソースに対して直接 IAM アクセス許可を持っていなくても、そのデータソースのメトリクスをクエリできます。例えば、Amazon Managed Service for Prometheus データソース Lambda 関数に対する `lambda:InvokeFunction` アクセス許可をユーザーに付与した場合、そのユーザーは対応する Amazon Managed Service for Prometheus ワークスペースに対して直接 IAM アクセスを持っていなくても、そのワークスペースからメトリクスをクエリできます。

データソースのテンプレート URL は、CloudWatch Settings Console の [スタックを作成] ページにあります。

```
{
  "Version":"2012-10-17",
  "Statement":[
    {
      "Effect" : "Allow",
      "Action" : [ "cloudformation:CreateStack" ],
      "Resource" : "*",
      "Condition" : {
        "StringEquals" : {
          "cloudformation:TemplateUrl" : [ data-source-template-url ]
```

```
        }
      }
    }
  ]
}
```

AWS CloudFormation アクセスを制御する方法の詳細については、「AWS Identity and Access Management によるアクセスの制御」を参照してください。

ウィザードによる事前構築済みのデータソースへの接続

このトピックでは、ウィザードを使用して CloudWatch を以下のデータソースに接続する手順について説明します。

- Amazon OpenSearch Service
- Amazon Managed Service for Prometheus
- Amazon RDS for MySQL
- Amazon RDS for PostgreSQL
- Amazon S3 CSV ファイル
- Microsoft Azure Monitor
- Prometheus

このセクションでは、後でいくつかのサブセクションに分けて、こうした各データソースを管理およびクエリする際の注意事項を示します。

データソースへのコネクタを作成するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで [設定] を選択します。
3. [メトリクスのデータソース] タブを選択します。
4. [データソースを作成] を選択します。
5. 目的のソースを選択し、[次へ] を選択します。
6. データソースの名前を入力します。
7. 選択したデータソースに応じて、その他の必要な情報を入力します。例えば、データソースにアクセスするための認証情報が必要になることもあれば、Prometheus ワークスペース名、デー

タベース名、Amazon S3 バケット名といったデータソース識別情報が必要になることもあります。AWS サービスの場合、ウィザードがリソースを検出して選択ドロップダウンに入力します。

現在使用中のデータソースに関するその他の注意事項については、この手順に続く各セクションを参照してください。

8. CloudWatch を VPC 内のデータソースに接続するには、[VPC を使用] を選択し、使用する VPC を選択します。次に、サブネットとセキュリティグループを選択します。
9. [AWS CloudFormation が IAM リソースを作成することを承認します] を選択します。このリソースは Lambda 関数実行ロールです。
10. [データソースを作成] を選択します。

    先ほど追加した新しいソースは、AWS CloudFormation スタックでの作成が完了するまで表示されません。進行状況を確認するには、[CloudFormation スタックのステータスを見る] を選択します。あるいは、更新アイコンを選択して、このリストを更新することもできます。

    新しいデータソースがこのリストに表示されたら、使用する準備ができたことになります。[CloudWatch メトリクスからクエリ] を選択して、クエリを開始できます。詳細については、「 別のデータソースにあるメトリクスのグラフ化」を参照してください。

Amazon Managed Service for Prometheus

データソース設定の更新

- データソースを手動で更新する場合は、次の操作を実行します。
  - Amazon Managed Service for Prometheus ワークスペース ID を更新するには、データソースコネクタ Lambda 関数の `AMAZON_PROMETHEUS_WORKSPACE_ID` 環境変数を更新します。
  - VPC 設定を更新する場合は、「VPC アクセスの設定 (コンソール)」で詳細を確認してください。

データソースのクエリ

- Amazon Managed Service for Prometheus をクエリする場合は、[マルチソースクエリ] タブでデータソースを選択し、Amazon Managed Service for Prometheus コネクタを選択した後で、クエリヘルパーを使用してメトリクスとラベルを検出し、簡単な PromQL クエリを実行できます。また、PromQL クエリエディタを使用して PromQL クエリを作成することもできます。

- CloudWatch データソースコネクタでは、複数行にわたるクエリはサポートされていません。そうしたクエリを実行するか、そうしたクエリでアラームやダッシュボードウィジェットを作成すると、すべてのラインフィードがスペースに置き換えられます。場合によっては、クエリが無効になることもあります。例えば、クエリに 1 行のコメントが含まれていると、そのクエリは無効になります。コマンドラインまたは Infrastructure as Code から複数行にわたるクエリを使用してダッシュボードまたはアラームを作成しようとすると、API がそのアクションを拒否して、解析エラーが発生します。

Amazon OpenSearch Service

データソースの作成

OpenSearch ドメインが FGAC で有効になっている場合、コネクタ Lambda 関数の実行ロールを OpenSearch Service 内のユーザーにマッピングする必要があります。詳細については、OpenSearch Service ドキュメントの「Managing permissions」の「Mapping users to roles」セクションを参照してください。

OpenSearch ドメインが Virtual Private Cloud (VPC) 内のみでアクセス可能である場合は、`AMAZON_OPENSEARCH_ENDPOINT` と呼ばれる Lambda 関数に新しい環境変数を手動で追加する必要があります。この変数の値は、OpenSearch エンドポイントのルートドメインにする必要があります。このルートドメインは、OpenSearch Service コンソールに一覧表示されているドメインエンドポイントから `https://` と `<region>.es.amazonaws.com` を削除することで取得できます。例えば、ドメインエンドポイントが `https://sample-domain.us-east-1.es.amazonaws.com` の場合、ルートドメインは `sample-domain` になります。

データソースの更新

- データソースを手動で更新する場合は、次の操作を実行します。
  - OpenSearch Service ドメインを更新するには、データソースコネクタ Lambda 関数の `AMAZON_OPENSEARCH_DOMAIN_NAME` 環境変数を更新します。
  - VPC 設定を更新する場合は、「VPC アクセスの設定 (コンソール)」で詳細を確認してください。

データソースのクエリ

- OpenSearch Service をクエリする場合は、[マルチソースクエリ] タブでデータソースを選択した後で、以下の操作を実行します。

- クエリするインデックスを選択します。
- メトリクス名 (ドキュメント内の任意の数値フィールド) と統計を選択します。
- 時間軸 (ドキュメント内の任意の日付フィールド) を選択します。
- 適用するフィルター (ドキュメント内の任意の文字列フィールド) を選択します。
- [グラフクエリ] を選択します。

Amazon RDS for PostgreSQL と Amazon RDS for MySQL

データソースの作成

- データソースが VPC 内でのみアクセス可能な場合は、「ウィザードによる事前構築済みのデータソースへの接続」で説明しているように、コネクタの VPC 設定を含める必要があります。認証情報を取得するためにデータソースを VPC に接続しなければならない場合は、VPC にエンドポイントを設定する必要があります。詳細については、「AWS Secrets Manager VPC エンドポイントの使用」を参照してください。

  さらに、Amazon RDS サービスの VPC エンドポイントを作成する必要があります。詳細については、「Amazon RDS API とインターフェイス VPC エンドポイント (AWS PrivateLink)」を参照してください。

データソースの更新

- データソースを手動で更新する場合は、次の操作を実行します。
  - データベースインスタンスを更新するには、データソースコネクタ Lambda 関数の `RDS_INSTANCE` 環境変数を更新します。
  - Amazon RDS への接続に使用するユーザー名とパスワードを更新するには、AWS Secrets Manager を使用します。データソースに使用されるシークレットの ARN は、データソース Lambda 関数の環境変数 `RDS_SECRET` で確認できます。AWS Secrets Manager でシークレットを更新する方法の詳細については、「Modify an AWS Secrets Manager secret」を参照してください。
  - VPC 設定を更新する場合は、「VPC アクセスの設定 (コンソール)」で詳細を確認してください。

データソースのクエリ

- Amazon RDS をクエリする場合は、[マルチソースクエリ] タブでデータソースを選択し、Amazon RDS コネクタを選択した後で、データベースディスカバラーを使用して、使用可能なデータベース、テーブル、列を表示できます。また、SQL エディタを使用して、SQL クエリを作成することもできます。

  作成した SQL クエリでは、以下の変数を使用できます。

  - `$start.iso` - ISO 日付形式の開始時刻
  - `$end.iso` - ISO 日付形式の終了時刻
  - `$period` - 選択した期間 (秒)

  例えば、`SELECT value, timestamp FROM table WHERE timestamp BETWEEN $start.iso and $end.iso` というクエリを実行できます。
- CloudWatch データソースコネクタでは、複数行にわたるクエリはサポートされていません。そうしたクエリを実行するか、そうしたクエリでアラームやダッシュボードウィジェットを作成すると、すべてのラインフィードがスペースに置き換えられます。場合によっては、クエリが無効になることもあります。例えば、クエリに 1 行のコメントが含まれていると、そのクエリは無効になります。コマンドラインまたは Infrastructure as Code から複数行にわたるクエリを使用してダッシュボードまたはアラームを作成しようとすると、API がそのアクションを拒否して、解析エラーが発生します。

> **Note**
>
> 結果に日付フィールドがない場合は、各数値フィールドの値が合計されて 1 つの値になり、指定された時間範囲にわたってプロットされます。タイムスタンプが CloudWatch で選択された期間と一致しない場合、データは `SUM` を使用して自動的に集計され、CloudWatch の期間に合わせて調整されます。

Amazon S3 CSV ファイル

データソースのクエリ

- Amazon S3 CSV ファイルをクエリする場合は、[マルチソースクエリ] タブでデータソースを選択し、Amazon S3 コネクタを選択した後で、Amazon S3 バケットとキーを選択します。

  CSV ファイルの形式は、次のようにする必要があります。
  - タイムスタンプは最初の列である必要があります。

- テーブルにはヘッダー行が必要です。ヘッダーは、メトリクスに名前を付けるために使用されます。タイムスタンプ列のタイトルは無視され、メトリクス列のタイトルのみが使用されます。
- タイムスタンプは ISO 日付形式である必要があります。
- メトリクスは数値フィールドである必要があります。

```
Timestamp, Metric-1, Metric-2, ...
```

以下に例を示します。

timestamp	CPU (%)	メモリ (%)	ストレージ (%)
2023-11-23T17:09:41+00:00	1	2	3
2023-11-23T17:04:41+00:00	4	5	6
2023-11-23T16:59:41+00:00	7	8	9
2023-11-23T16:54:41+00:00	10	11	12

> **Note**
>
> タイムスタンプが指定されていない場合は、各メトリクスの値が合計されて 1 つの値になり、指定された時間範囲にわたってプロットされます。タイムスタンプが CloudWatch で選択された期間と一致しない場合、データは SUM を使用して自動的に集計され、CloudWatch の期間に合わせて調整されます。

Microsoft Azure Monitor

データソースの作成

- Microsoft Azure Monitor に接続するには、テナント ID、クライアント ID、クライアントシークレットを指定する必要があります。認証情報は、AWS Secrets Manager に保存されます。詳細に

ついては、Microsoft のドキュメントの「Create a Microsoft Entra application and service principal that can access resources」を参照してください。

データソースの更新

- データソースを手動で更新する場合は、次の操作を実行します。
  - Azure Monitor への接続に使用されるテナント ID、クライアント ID、クライアントシークレットを更新するには、データソースに使用されるシークレットの ARN を確認します。これは、データソース Lambda 関数で AZURE_CLIENT_SECRET 環境変数として使用されています。AWS Secrets Manager でシークレットを更新する方法の詳細については、「Modify an AWS Secrets Manager secret」を参照してください。

データソースのクエリ

- Azure Monitor をクエリする場合は、[マルチソースクエリ] タブでデータソースを選択し、Azure Monitor コネクタを選択した後で、Azure のサブスクリプション、リソースグループ、リソースを指定します。次に、メトリクス名前空間、メトリクス、集計を選択し、ディメンションでフィルタリングします。

Prometheus

データソースの作成

- Prometheus エンドポイントのほか、Prometheus をクエリするために必要なユーザーとパスワードを指定する必要があります。認証情報は、AWS Secrets Manager に保存されます。
- データソースが VPC 内でのみアクセス可能な場合は、「 ウィザードによる事前構築済みのデータソースへの接続」で説明しているように、コネクタの VPC 設定を含める必要があります。認証情報を取得するためにデータソースを接続しなければならない場合は、VPC にエンドポイントを設定する必要があります。詳細については、「AWS Secrets Manager VPC エンドポイントの使用」を参照してください。

データソース設定の更新

- データソースを手動で更新する場合は、次の操作を実行します。
  - Prometheus エンドポイントを更新するには、データソース Lambda 関数で PROMETHEUS_API_ENDPOINT 環境変数として新しいエンドポイントを指定します。

- Prometheus への接続に使用されるユーザー名とパスワードを更新するには、データソースに使用されるシークレットの ARN を確認します。これは、データソース Lambda 関数で PROMETHEUS_API_SECRET 環境変数として使用されています。AWS Secrets Manager でシークレットを更新する方法の詳細については、「Modify an AWS Secrets Manager secret」を参照してください。
- VPC 設定を更新する場合は、「VPC アクセスの設定 (コンソール)」で詳細を確認してください。

データソースのクエリ

> **Important**
>
> Prometheus メトリクスタイプは CloudWatch メトリクスとは異なり、Prometheus で使用できるメトリクスの多くはその仕様上累積的です。Prometheus メトリクスをクエリするときに、データに追加で何らかの変換が適用されることはありません。メトリクス名またはラベルのみを指定した場合は、累積された値が表示されます。詳細については、Prometheus ドキュメントの「Metric types」を参照してください。
> Prometheus メトリクスデータを CloudWatch メトリクスのように個別の値として表示するには、クエリをその実行前に編集する必要があります。例えば、場合によっては Prometheus メトリクス名よりも先に rate 関数への呼び出しを追加する必要があります。rate 関数とその他の Prometheus 関数のドキュメントについては、Prometheus ドキュメントの「rate()」を参照してください。

CloudWatch データソースコネクタでは、複数行にわたるクエリはサポートされていません。そうしたクエリを実行するか、そうしたクエリでアラームやダッシュボードウィジェットを作成すると、すべてのラインフィードがスペースに置き換えられます。場合によっては、クエリが無効になることもあります。例えば、クエリに 1 行のコメントが含まれていると、そのクエリは無効になります。コマンドラインまたは Infrastructure as Code から複数行にわたるクエリを使用してダッシュボードまたはアラームを作成しようとすると、API がそのアクションを拒否して、解析エラーが発生します。

使用可能な更新の通知

Amazon から随時、新たに使用可能になったバージョンでコネクタを更新することをお勧めする旨が通知されます。その際、更新手順も一緒に通知されます。

データソースへのカスタムコネクタの作成

カスタムデータソースを CloudWatch に接続する場合、次の 2 つの選択肢があります。

- CloudWatch に用意されているサンプルのテンプレートを使用することから始めます。このテンプレートでは、JavaScript または Python のいずれかを使用できます。テンプレートには、Lambda 関数を作成するときに便利なサンプルの Lambda コードが含まれています。次に、カスタムデータソースに接続するようにテンプレートの Lambda 関数を変更できます。
- CloudWatch で使用するデータソースコネクタ、データクエリ、時系列の準備を実装する AWS Lambda 関数をゼロから作成します。具体的には、データポイントを必要に応じて事前集約またはマージし、CloudWatch との互換性が確保されるように期間とタイムスタンプを調整するという関数にします。

目次

テンプレートの使用

テンプレートを使用すると、サンプルの Lambda 関数が作成されるので、カスタムコネクタをすばやく構築できます。こうしたサンプル関数には、カスタムコネクタの構築でよくあるシナリオに対応したサンプルコードが用意されています。テンプレートを使用してコネクタを作成したら、Lambda コードを開き、そのコネクタをデータソースへの接続に使用するようにコードを変更できます。

また、テンプレートを使用した場合、CloudWatch が Lambda アクセス許可ポリシーの作成と Lambda 関数へのリソースタグのアタッチを行います。

テンプレートを使用してカスタムデータソースへのコネクタを作成するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで [設定] を選択します。
3. [メトリクスのデータソース] タブを選択します。
4. [データソースを作成] を選択します。
5. [カスタム - 入門ガイドテンプレート] のラジオボタンを選択し、[次へ] を選択します。
6. データソースの名前を入力します。
7. リストされているテンプレートのいずれかを選択します。
8. Node.js または Python を選択します。
9. [データソースを作成] を選択します。

   先ほど追加した新しいカスタムソースは、AWS CloudFormation スタックでの作成が完了するまで表示されません。進行状況を確認するには、[CloudFormation スタックのステータスを見る] を選択します。あるいは、更新アイコンを選択して、このリストを更新することもできます。

   新しいデータソースがこのリストに表示されたら、コンソールでテストして変更できます。
10. (オプション) コンソールでこのソースのテストデータをクエリする場合は、「別のデータソースにあるメトリクスのグラフ化」の手順に従ってください。
11. 必要に応じて Lambda 関数を変更します。

    a. ナビゲーションペインで [設定] を選択します。

    b. [メトリクスのデータソース] タブを選択します。

    c. 変更するソースに対して [Lambda コンソールで表示] を選択します。

    これで、データソースにアクセスするように関数を変更できます。詳細については、「ステップ 1: 関数を作成する」を参照してください。

    > **Note**
    >
    > テンプレートを使用すると、Lambda 関数を記述する際に「ステップ 2: Lambda アクセス許可ポリシーを作成する」や「ステップ 3: シークレットリソースを Lambda 関数にアタッチする」の手順に従う必要がありません。テンプレートを使用したため、そのいずれのステップも CloudWatch によって実行されるからです。

ゼロからのカスタムデータソースの作成

このセクションのステップに従って、CloudWatch をデータソースに接続する Lambda 関数を作成します。

ステップ 1: 関数を作成する

カスタムデータソースコネクタは、CloudWatch の `GetMetricData` イベントをサポートする必要があります。必要に応じて、`DescribeGetMetricData` イベントを実装して、CloudWatch コンソールでユーザーにコネクタの使用方法に関するドキュメントを提供することもできます。また、`DescribeGetMetricData` レスポンスを使用して、CloudWatch カスタムクエリビルダーに使用されるデフォルトを設定することもできます。

CloudWatch には、最初の一歩に役立つサンプルとしてコードスニペットが用意されています。詳細については、https://github.com/aws-samples/cloudwatch-data-source-samples でサンプルリポジトリを参照してください。

制約

- Lambda からのレスポンスは 6 MB 未満でなければなりません。レスポンスが 6 MB を超える場合、`GetMetricData` レスポンスは Lambda 関数を `InternalError` としてマークし、データは返されません。
- Lambda 関数は、可視化とダッシュボードの用途では 10 秒以内に、アラームの用途では 4.5 秒以内に実行を完了する必要があります。実行時間がその時間を超えた場合、`GetMetricData` レスポンスは Lambda 関数を `InternalError` としてマークし、データは返されません。
- Lambda 関数は、エポックタイムスタンプを秒単位で使用して、その出力を送信する必要があります。
- Lambda 関数がデータを再サンプリングするのではなく CloudWatch ユーザーがリクエストした開始時間と期間の長さに対応しないデータを返した場合、CloudWatch ではそのデータは無視されます。こうした余分なデータは、いずれの可視化やアラームからも破棄されます。開始時刻から終了時刻までの範囲内にないデータも破棄されます。

  例えば、ユーザーが 10:00 から 11:00 まで 5 分間隔でデータを要求した場合、「10:00:00 から 10:04:59 まで」と「10:05:00 から 10:09:59 まで」は有効な時間範囲としてデータが返されます。`10:00 value1`、`10:05 value2` といった時系列を返す必要があります。例えば、関数から `10:03 valueX` が返された場合、10:03 はリクエストされた開始時間と期間に対応していないため、関数はドロップされます。

- CloudWatch データソースコネクタでは、複数行にわたるクエリはサポートされていません。そうしたクエリを実行するか、そうしたクエリでアラームやダッシュボードウィジェットを作成すると、すべてのラインフィードがスペースに置き換えられます。場合によっては、クエリが無効になることもあります。

GetMetricData イベント

リクエストペイロード

次に、Lambda 関数への入力として送信される GetMetricData リクエストペイロードの例を示します。

```
{
  "EventType": "GetMetricData",
  "GetMetricDataRequest": {
    "StartTime": 1697060700,
    "EndTime": 1697061600,
    "Period": 300,
    "Arguments": ["serviceregistry_external_http_requests{host_cluster!=\"prod\"}"]
  }
}
```

- StartTime - 返されるデータの中で最も古いデータであることを指定するタイムスタンプです。Type は、タイムスタンプエポック秒です。
- EndTime - 返されるデータの中で最も新しいデータであることを指定するタイムスタンプです。Type は、タイムスタンプエポック秒です。
- Period — メトリクスデータの各集計が表す秒数です。最小値は 60 秒です。Type は秒です。
- Arguments - Lambda メトリクス数式に渡す一連の引数です。引数を渡す方法の詳細については、「Lambda 関数に引数を渡す方法」を参照してください。

レスポンスペイロード

次に、Lambda 関数から返される GetMetricData レスポンスペイロードの例を示します。

```
{
   "MetricDataResults": [
      {
         "StatusCode": "Complete",
```

```
            "Label": "CPUUtilization",
            "Timestamps": [ 1697060700, 1697061000, 1697061300 ],
            "Values": [ 15000, 14000, 16000 ]
        }
    ]
}
```

レスポンスペイロードには、MetricDataResults フィールドまたは Error フィールドのいずれかが含まれ、両方が含まれることはありません。

MetricDataResults フィールドは、タイプ MetricDataResult の時系列フィールドのリストです。こうした時系列フィールドごとに、以下のフィールドを含めることができます。

- StatusCode - (オプション) Complete は、リクエストした時間範囲内のデータポイントがすべて返されたことを示します。PartialData は、データポイントが一部不足した状態で返されたことを示します。これを省略した場合、デフォルトは Complete です。

  有効な値: Complete | InternalError | PartialData | Forbidden
- Messages - オプションのメッセージリストです。どのようなデータが返されたかに関する情報が追加で含まれています。

  Type: Code と Value の文字列が含まれている MessageData オブジェクトの配列です。
- Label - データに関連付けられている判読可能なラベルです。

  型: 文字列
- Timestamps - エポック時間でフォーマットされた、データポイントのタイムスタンプです。タイムスタンプの数は常に値の数に一致し、Timestamps[x] の値は Values[x] です。

  Type: タイムスタンプの配列
- Values - メトリクスのデータポイントで、Timestamps に対応しています。値の数は常にタイムスタンプの数に一致し、Timestamps[x] の値は Values[x] です。

  Type: 倍精度の配列

Error オブジェクトの詳細については、以下のセクションを参照してください。

エラーレスポンス形式

必要に応じて、エラーレスポンスを使用してエラーの詳しい情報を提供できます。パラメータの欠落やパラメータの型の誤りなど検証エラーが発生した場合には、コード検証を使用してエラーを返すことをお勧めします。

次に、Lambda 関数で GetMetricData 検証例外が発生した場合のレスポンスの例を示します。

```
{
   "Error": {
      "Code": "Validation",
      "Value": "Invalid Prometheus cluster"
   }
}
```

次に、アクセスの問題があるために Lambda 関数がデータを返すことができない場合のレスポンスの例を示します。レスポンスは単一の時系列に変換され、Forbidden というステータスコードが返されます。

```
{
   "Error": {
      "Code": "Forbidden",
      "Value": "Unable to access ..."
   }
}
```

次に、Lambda 関数で全体的な InternalError 例外が発生した場合のレスポンスの例を示します。レスポンスは単一の時系列に変換され、InternalError というステータスコードとメッセージが返されます。エラーコードに Validation や Forbidden 以外の値があるたびに、CloudWatch では汎用的な内部エラーが発生したと見なされます。

```
{
   "Error": {
      "Code": "PrometheusClusterUnreachable",
      "Value": "Unable to communicate with the cluster"
   }
}
```

DescribeGetMetricData イベント

リクエストペイロード

次に、DescribeGetMetricData リクエストペイロードの例を示します。

```
{
  "EventType": "DescribeGetMetricData"
}
```

レスポンスペイロード

次に、DescribeGetMetricData レスポンスペイロードの例を示します。

```
{
    "Description": "Data source connector",
    "ArgumentDefaults": [{
        Value: "default value"
     }]
}
```

- Description - データソースコネクタの使用方法の説明です。この説明は、CloudWatch コンソールに表示されます。Markdown がサポートされています。

  型: 文字列
- ArgumentDefaults - カスタムデータソースビルダーを事前に入力しておくために必要に応じて使用できる、引数のデフォルト値からなる配列です。

  [{ Value: "default value 1"}, { Value: 10}] が返された場合、CloudWatch コンソールのクエリビルダーに入力が 2 つ表示されます。1 つ目は「デフォルト値 1」で、2 つ目は 10 です。

  ArgumentDefaults を指定しない場合、入力は 1 つだけ表示され、型がデフォルトの String に設定されます。

  Type: 値と型が含まれているオブジェクトの配列です。
- Error - (オプション) エラーフィールドは、どのレスポンスにも含めることができます。「GetMetricData イベント」で例を確認できます。

CloudWatch アラームに関する重要な考慮事項

データソースを使用して CloudWatch アラームを設定する場合は、データをタイムスタンプ付きで 1 分ごとに CloudWatch に報告するように設定する必要があります。接続先のデータソースのメトリクスに対してアラームを作成する方法の詳細とその他の考慮事項については、「接続されたデータソースに基づいてアラームを作成する」を参照してください。

(オプション) AWS Secrets Manager を使用した認証情報の保存

Lambda 関数で認証情報を使用してデータソースにアクセスする必要がある場合は、認証情報を Lambda 関数にハードコーディングするのではなく、AWS Secrets Manager を使用して認証情報を保存しておくことをお勧めします。Lambda で AWS Secrets Manager を使用する方法の詳細については、「Use AWS Secrets Manager secrets in AWS Lambda functions」を参照してください。

(オプション) VPC のデータソースへの接続

データソースが Amazon Virtual Private Cloud によって管理される VPC にある場合は、そのデータソースにアクセスするように Lambda 関数を設定する必要があります。詳細については、「アウトバウンドネットワークを VPC 内のリソースに接続する」を参照してください。

場合によっては、AWS Secrets Manager などのサービスにアクセスするように VPC サービスエンドポイントを設定する必要があります。詳細については、「インターフェイス VPC エンドポイントを使用して AWS サービスにアクセスする」を参照してください。

ステップ 2: Lambda アクセス許可ポリシーを作成する

作成した Lambda 関数を使用するためには、ポリシーステートメントを作成して必要な CloudWatch アクセス許可を付与する必要があります。ポリシーステートメントは、AWS CLI または Lambda コンソールを使用して作成できます。

AWS CLI を使用してポリシーステートメントを作成するには

- 次のコマンドを入力します。*123456789012* を自分のアカウント ID に、*my-data-source-function* を Lambda 関数の名前に、*MyDataSource-DataSourcePermission1234* を任意の一意の値にそれぞれ置き換えます。

```
aws lambda add-permission --function-name my-data-source-function --statement-
id MyDataSource-DataSourcePermission1234 --action lambda:InvokeFunction --principal
 lambda.datasource.cloudwatch.amazonaws.com --source-account 123456789012
```

ステップ 3: シークレットリソースを Lambda 関数にアタッチする

CloudWatch コンソールが、タグを使用してどの Lambda 関数がデータソースコネクタであるかを判断します。いずれかのウィザードを使用してデータソースを作成すると、その設定を行う AWS CloudFormation スタックによってタグが自動的に適用されます。データソースを自分で作成する場合は、Lambda 関数に以下のタグを使用できます。これにより、メトリクスをクエリしたときに、コネクタが CloudWatch コンソールの [データソース] ドロップダウンに表示されます。

- `cloudwatch:datasource` をキーとし、`custom` を値とするタグ。

カスタムデータソースの使用

データソースを作成したら、そのデータソースのデータをクエリして可視化し、アラームを設定できます。テンプレートを使用してカスタムデータソースコネクタを作成した場合や、 ステップ 3: シークレットリソースを Lambda 関数にアタッチする にリストされているタグを追加した場合は、「 別のデータソースにあるメトリクスのグラフ化」の手順に従ってデータをクエリできます。

また、この後のセクションで説明しているように、メトリクス数学関数 LAMBDA を使用してデータをクエリすることもできます。

データソースのメトリクスに対してアラームを作成する方法については、「接続されたデータソースに基づいてアラームを作成する」を参照してください。

Lambda 関数に引数を渡す方法

カスタムデータソースに引数を渡す場合は、データソースをクエリするときに CloudWatch コンソールのクエリビルダーを使用するという方法をお勧めします。

また、CloudWatch メトリクス数式の新しい LAMBDA 式を使用することで、Lambda 関数を使用してデータソースからデータを取得することもできます。

```
LAMBDA("LambdaFunctionName" [, optional-arg]*)
```

`optional-arg` は、最大 20 個の文字列、数値、またはブール値です。例えば、`param`、`3.14`、または `true` などです。

> **Note**
>
> CloudWatch データソースコネクタでは、複数行にわたる文字列はサポートされていません。そうしたクエリを実行するか、そうしたクエリでアラームやダッシュボードウィジェットを作成すると、すべてのラインフィードがスペースに置き換えられます。場合によっては、クエリが無効になることもあります。

LAMBDA メトリクス数学関数を使用する場合は、関数名 ("MyFunction") を指定できます。リソースポリシーで許可されている場合は、関数の特定のバージョン ("MyFunction:22") や Lambda 関

数のエイリアス ("MyFunction:MyAlias") を使用することもできます。* を使用することはできません。

次に、LAMBDA 関数の呼び出し例をいくつか示します。

```
LAMBDA("AmazonOpenSearchDataSource", "MyDomain", "some-query")
```

```
LAMBDA("MyCustomDataSource", true, "fuzzy", 99.9)
```

LAMBDA メトリクス数学関数は時系列のリストを返します。そのリストをリクエストに返したり、他のメトリクス数学関数と組み合わせて使用したりできます。次に、LAMBDA を他のメトリクス数学関数と組み合わせた場合の例を示します。

```
FILL(LAMBDA("AmazonOpenSearchDataSource", "MyDomain", "some-query"), 0)
```

データソースへのコネクタの削除

データソースへのコネクタを削除するには、このセクションで示す手順に従ってください。

データソースへのコネクタを削除するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで [設定] を選択します。
3. [メトリクスのデータソース] タブを選択します。
4. 削除するデータソースの行で [CloudFormation で管理] を選択します。

   AWS CloudFormation コンソールに自動的に移動します。
5. データソースの名前が付いているセクションで、[削除] を選択します。
6. 確認ポップアップで、[削除] を選択します。

CloudWatch エージェントを使用してメトリクス、ログ、トレースを収集する

統合 CloudWatch エージェントを使用すると、以下のことを実行できます。

- オペレーティングシステム全体で Amazon EC2 インスタンスから内部システムレベルのメトリクスを収集します。このメトリクスには、EC2 インスタンスのメトリクスに加えて、ゲスト内メトリクスを含めることができます。収集することができる追加のメトリクスについては、「CloudWatch エージェントにより収集されるメトリクス」を参照してください。
- オンプレミスサーバーからシステムレベルのメトリクスを収集します。これには、ハイブリッド環境のサーバーや AWS によって管理されていないサーバーも含まれる可能性があります。
- カスタムメトリクスは、StatsD および collectd プロトコルを使用して、アプリケーションまたはサービスから取得します。StatsD は、Linux サーバーと、Windows Server を実行するサーバーの両方でサポートされています。collectd は、Linux サーバーでのみサポートされています。
- Linux または Windows Server を実行している Amazon EC2 インスタンスおよびオンプレミスサーバーから、ログを収集します。

  > **Note**
  >
  > CloudWatch エージェントは、FIFO パイプからのログの収集をサポートしていません。

- バージョン 1.300031.0 以降を使用すると、CloudWatch Application Signals が使用可能になります。詳細については、「Application Signals」を参照してください。
- バージョン 1.300025.0 以降では、OpenTelemetry または X-Ray クライアント SDK からトレースを収集し、それらを X-Ray に送信できます。

  CloudWatch エージェントを使用すると、別途トレース収集デーモンを実行しなくてもトレースを収集できるため、実行および管理するエージェントの数を減らすことができます。

CloudWatch エージェントで収集したメトリクスは、他の CloudWatch メトリクスと同様に、CloudWatch でも保存して表示できます。CloudWatch エージェントにより収集されるメトリクスのデフォルトの名前空間は CWAgent ですが、エージェントを構成するときに別の名前空間を指定できます。

統合 CloudWatch エージェントによって収集されたログは、古い CloudWatch Logs エージェントによって収集されたログと同様に、Amazon CloudWatch Logs に処理され、保存されます。CloudWatch Logs の料金の詳細については、Amazon CloudWatch の料金をご覧ください。

CloudWatch エージェントによって収集されたメトリクスは、カスタムメトリクスとして請求されます。CloudWatch メトリクスの料金の詳細については、「Amazon CloudWatch の料金」を参照してください。

CloudWatch エージェントは MIT ライセンスの下でオープンソースであり、GitHub でホストされています。CloudWatch エージェントの構築、カスタマイズ、または貢献をご希望の場合は、最新の手順について GitHub リポジトリを参照してください。セキュリティの問題の兆候を見つけても、GitHub などの公開フォーラムに投稿しないでください。代わりに、脆弱性レポートまたは AWS セキュリティに直接 E メールを送信するの手順に従ってください。

このセクションのステップでは、Amazon EC2 インスタンスとオンプレミスサーバーに統合 CloudWatch エージェントをインストールする方法について説明します。CloudWatch エージェントで収集できるメトリクスの詳細については、「CloudWatch エージェントにより収集されるメトリクス」を参照してください。

サポートされるオペレーティングシステム

CloudWatch エージェントは、以下のオペレーティングシステム上の x86-64 アーキテクチャでサポートされています。また、ここに記載されている各メジャーバージョンのすべてのマイナーバージョンアップデートでもサポートされています。

- Amazon Linux 2023
- Amazon Linux 2
- Ubuntu Server バージョン 23.10、22.04、20.04、18.04、16.04、14.04
- CentOS バージョン 9、8、および 7
- Red Hat Enterprise Linux (RHEL) バージョン 9、8、および 7
- Debian バージョン 12、11、10
- SUSE Linux Enterprise Server (SLES) バージョン 15 および 12
- Oracle Linux バージョン 9、8、7
- AlmaLinux バージョン 9 および 8
- Rocky Linux バージョン 9 および 8
- 次の macOS コンピュータ: EC2 M1 Mac1 インスタンスおよび macOS 14 (Sonoma)、macOS 13 (Ventura)、macOS 12 (Monterey) で動作しているコンピュータ

- 64 ビットバージョンの Windows Server 2022、Windows Server 2019、Windows Server 2016
- 64 ビット Windows 10

エージェントは、以下のオペレーティングシステム上の ARM64 アーキテクチャでサポートされています。また、ここに記載されている各メジャーバージョンのすべてのマイナーバージョンアップデートでもサポートされています。

- Amazon Linux 2023
- Amazon Linux 2
- Ubuntu Server バージョン 23.10、22.04、20.04、18.04、16.04
- CentOS バージョン 9 および 8
- Red Hat Enterprise Linux (RHEL) バージョン 9、8、および 7
- Debian バージョン 12、11、10
- SUSE Linux Enterprise Server 15
- 次の macOS コンピュータ: macOS 14 (Sonoma)、macOS 13 (Ventura)、macOS 12 (Monterey)

インストールプロセスの概要

CloudWatch エージェントは、コマンドラインを使用して手動でダウンロードおよびインストールできます。または、SSM と統合することもできます。いずれかのメソッドを使用して CloudWatch エージェントをインストールする一般的な流れは次のとおりです。

1. IAM ロールまたはユーザーを作成し、エージェントでサーバーからメトリクスを収集するか、必要に応じて と統合できるようにしますAWS Systems Manager
2. エージェントパッケージをダウンロードします。
3. CloudWatch エージェント設定ファイルを変更して、収集するメトリクスを指定します。
4. サーバーにエージェントをインストールして起動します。EC2 インスタンスにエージェントをインストールする際、ステップ 1 で作成した IAM ロールをアタッチします。オンプレミスサーバー上にエージェントをインストールしたら、ステップ 1 で作成した IAM ユーザーの認証情報を含む名前付きプロファイルを指定します。

内容

- Amazon CloudWatch Observability EKS アドオンを使用して CloudWatch エージェントをインストールする
- CloudWatch エージェントにより収集されるメトリクス
- CloudWatch エージェントの一般的なシナリオ
- CloudWatch エージェントのトラブルシューティング

CloudWatch エージェントのインストール

CloudWatch エージェントは、Amazon Linux 2023 および Amazon Linux 2 でパッケージとして利用できます。これらのオペレーティング システムのいずれかを使用している場合は、以下のコマンドを入力してパッケージをインストールできます。また、インスタンスにアタッチされた IAM ロールに CloudWatchAgentServerPolicy がアタッチされていることを確認する必要もあります。詳細については、「Amazon EC2 インスタンスの CloudWatch エージェントで使用する IAM ロールを作成する」を参照してください。

```
sudo yum install amazon-cloudwatch-agent
```

Linux や Windows Server など、サポートされているすべてのオペレーティングシステムで、コマンドラインに Amazon S3 ダウンロードリンクを使用するか、Amazon EC2 Systems Manager を使用するか、AWS CloudFormation テンプレートを使用するかのいずれかで CloudWatch エージェントをダウンロードしてインストールできます。詳細については、以下のセクションを参照してください。

内容

- コマンドラインを使用した CloudWatch エージェントのインストール
- AWS Systems Manager を使用した CloudWatch エージェントのインストール
- AWS CloudFormation を使用して新しいインスタンスに CloudWatch エージェントをインストールする
- CloudWatch エージェント認証情報の優先設定
- CloudWatch エージェントパッケージの署名の検証

コマンドラインを使用した CloudWatch エージェントのインストール

以下のトピックを使用して、CloudWatch エージェントパッケージのダウンロード、設定、インストールを行います。

トピック

コマンドラインを使用して CloudWatch エージェントをダウンロードおよび設定する

以下のステップを使用して、CloudWatch エージェントパッケージをダウンロードし、IAM ロールまたはユーザーを作成します。また、必要に応じて、共通の設定ファイルを変更します。

CloudWatch エージェントパッケージをダウンロードする

> **Note**
>
> CloudWatch エージェントをダウンロードするには、接続に TLS 1.2 以降を使用する必要があります。

CloudWatch エージェントは、Amazon Linux 2023 および Amazon Linux 2 でパッケージとして利用できます。このオペレーティングシステムを使用している場合は、以下のコマンドを入力してパッケージをインストールできます。また、インスタンスにアタッチされた IAM ロールに CloudWatchAgentServerPolicy がアタッチされていることを確認する必要もあります。詳細については、「CloudWatch エージェントで使用する IAM ロールとユーザーを作成する」を参照してください。

```
sudo yum install amazon-cloudwatch-agent
```

サポートされているすべてのオペレーティングシステムで、コマンドラインを使用して、CloudWatch エージェントをダウンロードしてインストールできます。

ダウンロードリンクごとに、全般的なリンクと各リージョンのリンクがあります。例えば、Amazon Linux 2023、Amazon Linux 2、x86-64 アーキテクチャの場合、有効なダウンロードリンクは次の 3 つです。

- `https://amazoncloudwatch-agent.s3.amazonaws.com/amazon_linux/amd64/latest/amazon-cloudwatch-agent.rpm`
- `https://amazoncloudwatch-agent-us-east-1.s3.us-east-1.amazonaws.com/amazon_linux/amd64/latest/amazon-cloudwatch-agent.rpm`

- https://amazoncloudwatch-agent-eu-central-1.s3.eu-central-1.amazonaws.com/amazon_linux/amd64/latest/amazon-cloudwatch-agent.rpm

エージェントに対する最新の変更に関する README ファイル、およびダウンロード可能なバージョン番号を示すファイルをダウンロードすることもできます。これらのファイルは次の場所にあります。

- https://amazoncloudwatch-agent.s3.amazonaws.com/info/latest/RELEASE_NOTES 、、または https://amazoncloudwatch-agent-*region*.s3.*region*.amazonaws.com/info/latest/RELEASE_NOTES
- https://amazoncloudwatch-agent.s3.amazonaws.com/info/latest/CWAGENT_VERSION 、、または https://amazoncloudwatch-agent-*region*.s3.*region*.amazonaws.com/info/latest/CWAGENT_VERSION

アーキテクチャ	プラットフォーム	ダウンロードリンク	署名ファイルリンク
x86-64	Amazon Linux 2023 および Amazon Linux 2	https://amazoncloudwatch-agent.s3.amazonaws.com/amazon_linux/amd64/latest/amazon-cloudwatch-agent.rpm https://amazoncloudwatch-agent-*region*.s3.*region*.amazonaws.com/amazon_linux/amd64/latest/amazon-cloudwatch-agent.rpm	https://amazoncloudwatch-agent.s3.amazonaws.com/amazon_linux/amd64/latest/amazon-cloudwatch-agent.rpm.sig https://amazoncloudwatch-agent-*region*.s3.*region*.amazonaws.com/amazon_linux/amd64/latest/amazon-cloudwatch-agent.rpm.sig
x86-64	CentOS	https://amazoncloudwatch-agent.s3.amazonaws.com/centos/amd64/latest/amazon-cloudwatch-agent.rpm https://amazoncloudwatch-agent-*region*.s3.*region*.amazon	https://amazoncloudwatch-agent.s3.amazonaws.com/centos/amd64/latest/amazon-cloudwatch-agent.rpm.sig https://amazoncloudwatch-agent-*region*.s3.*region*.amazonaw

アーキテクチャ	プラットフォーム	ダウンロードリンク	署名ファイルリンク
		s.com/centos/amd64/latest/amazon-cloudwatch-agent.rpm	s.com/centos/amd64/latest/amazon-cloudwatch-agent.rpm.sig
x86-64	Redhat	https://amazoncloudwatch-agent.s3.amazonaws.com/redhat/amd64/latest/amazon-cloudwatch-agent.rpm https://amazoncloudwatch-agent-*region*.s3.*region*.amazonaws.com/redhat/amd64/latest/amazon-cloudwatch-agent.rpm	https://amazoncloudwatch-agent.s3.amazonaws.com/redhat/amd64/latest/amazon-cloudwatch-agent.rpm.sig https://amazoncloudwatch-agent-*region*.s3.*region*.amazonaws.com/redhat/amd64/latest/amazon-cloudwatch-agent.rpm.sig
x86-64	SUSE	https://amazoncloudwatch-agent.s3.amazonaws.com/suse/amd64/latest/amazon-cloudwatch-agent.rpm https://amazoncloudwatch-agent-*region*.s3.*region*.amazonaws.com/suse/amd64/latest/amazon-cloudwatch-agent.rpm	https://amazoncloudwatch-agent.s3.amazonaws.com/suse/amd64/latest/amazon-cloudwatch-agent.rpm.sig https://amazoncloudwatch-agent-*region*.s3.*region*.amazonaws.com/suse/amd64/latest/amazon-cloudwatch-agent.rpm.sig
x86-64	Debian	https://amazoncloudwatch-agent.s3.amazonaws.com/debian/amd64/latest/amazon-cloudwatch-agent.deb https://amazoncloudwatch-agent-*region*.s3.*region*.amazonaws.com/debian/amd64/latest/amazon-cloudwatch-agent.deb	https://amazoncloudwatch-agent.s3.amazonaws.com/debian/amd64/latest/amazon-cloudwatch-agent.deb.sig https://amazoncloudwatch-agent-*region*.s3.*region*.amazonaws.com/debian/amd64/latest/amazon-cloudwatch-agent.deb.sig

アーキテクチャ	プラットフォーム	ダウンロードリンク	署名ファイルリンク
x86-64	Ubuntu	https://amazoncloudwatch-agent.s3.amazonaws.com/ubuntu/amd64/latest/amazon-cloudwatch-agent.deb https://amazoncloudwatch-agent-*region*.s3.*region*.amazonaws.com/ubuntu/amd64/latest/amazon-cloudwatch-agent.deb	https://amazoncloudwatch-agent.s3.amazonaws.com/ubuntu/amd64/latest/amazon-cloudwatch-agent.deb.sig https://amazoncloudwatch-agent-*region*.s3.*region*.amazonaws.com/ubuntu/amd64/latest/amazon-cloudwatch-agent.deb.sig
x86-64	Oracle	https://amazoncloudwatch-agent.s3.amazonaws.com/oracle_linux/amd64/latest/amazon-cloudwatch-agent.rpm https://amazoncloudwatch-agent-*region*.s3.*region*.amazonaws.com/oracle_linux/amd64/latest/amazon-cloudwatch-agent.rpm	https://amazoncloudwatch-agent.s3.amazonaws.com/oracle_linux/amd64/latest/amazon-cloudwatch-agent.rpm.sig https://amazoncloudwatch-agent-*region*.s3.*region*.amazonaws.com/oracle_linux/amd64/latest/amazon-cloudwatch-agent.rpm.sig
x86-64	macOS	https://amazoncloudwatch-agent.s3.amazonaws.com/darwin/amd64/latest/amazon-cloudwatch-agent.pkg https://amazoncloudwatch-agent-*region*.s3.*region*.amazonaws.com/darwin/amd64/latest/amazon-cloudwatch-agent.pkg	https://amazoncloudwatch-agent.s3.amazonaws.com/darwin/amd64/latest/amazon-cloudwatch-agent.pkg.sig https://amazoncloudwatch-agent-*region*.s3.*region*.amazonaws.com/darwin/amd64/latest/amazon-cloudwatch-agent.pkg.sig

アーキテクチャ	プラットフォーム	ダウンロードリンク	署名ファイルリンク
x86-64	Windows	https://amazoncloudwatch-agent.s3.amazonaws.com/windows/amd64/latest/amazon-cloudwatch-agent.msi https://amazoncloudwatch-agent-*region*.s3.*region*.amazonaws.com/windows/amd64/latest/amazon-cloudwatch-agent.msi	https://amazoncloudwatch-agent.s3.amazonaws.com/windows/amd64/latest/amazon-cloudwatch-agent.msi.sig https://amazoncloudwatch-agent-*region*.s3.*region*.amazonaws.com/windows/amd64/latest/amazon-cloudwatch-agent.msi.sig
ARM64	Amazon Linux 2023 および Amazon Linux 2	https://amazoncloudwatch-agent.s3.amazonaws.com/amazon_linux/arm64/latest/amazon-cloudwatch-agent.rpm https://amazoncloudwatch-agent-*region*.s3.*region*.amazonaws.com/amazon_linux/arm64/latest/amazon-cloudwatch-agent.rpm	https://amazoncloudwatch-agent.s3.amazonaws.com/amazon_linux/arm64/latest/amazon-cloudwatch-agent.rpm.sig https://amazoncloudwatch-agent-*region*.s3.*region*.amazonaws.com/amazon_linux/arm64/latest/amazon-cloudwatch-agent.rpm.sig
ARM64	Red Hat	https://amazoncloudwatch-agent.s3.amazonaws.com/redhat/arm64/latest/amazon-cloudwatch-agent.rpm https://amazoncloudwatch-agent-*region*.s3.*region*.amazonaws.com/redhat/arm64/latest/amazon-cloudwatch-agent.rpm	https://amazoncloudwatch-agent.s3.amazonaws.com/redhat/arm64/latest/amazon-cloudwatch-agent.rpm.sig https://amazoncloudwatch-agent-*region*.s3.*region*.amazonaws.com/redhat/arm64/latest/amazon-cloudwatch-agent.rpm.sig

アーキテクチャ	プラットフォーム	ダウンロードリンク	署名ファイルリンク
ARM64	Ubuntu	https://amazoncloudwatch-agent.s3.amazonaws.com/ubuntu/arm64/latest/amazon-cloudwatch-agent.deb https://amazoncloudwatch-agent-*region*.s3.*region*.amazonaws.com/ubuntu/arm64/latest/amazon-cloudwatch-agent.deb	https://amazoncloudwatch-agent.s3.amazonaws.com/ubuntu/arm64/latest/amazon-cloudwatch-agent.deb.sig https://amazoncloudwatch-agent-*region*.s3.*region*.amazonaws.com/ubuntu/arm64/latest/amazon-cloudwatch-agent.deb.sig
ARM64	SUSE	https://amazoncloudwatch-agent.s3.amazonaws.com/suse/arm64/latest/amazon-cloudwatch-agent.rpm https://amazoncloudwatch-agent-*region*.s3.*region*.amazonaws.com/suse/arm64/latest/amazon-cloudwatch-agent.rpm	https://amazoncloudwatch-agent.s3.amazonaws.com/suse/arm64/latest/amazon-cloudwatch-agent.rpm.sig https://amazoncloudwatch-agent-*region*.s3.*region*.amazonaws.com/suse/arm64/latest/amazon-cloudwatch-agent.rpm.sig
ARM64	MacOS	https://amazoncloudwatch-agent.s3.amazonaws.com/darwin/arm64/latest/amazon-cloudwatch-agent.pkg https://amazoncloudwatch-agent-*region*.s3.*region*.amazonaws.com/darwin/arm64/latest/amazon-cloudwatch-agent.pkg	https://amazoncloudwatch-agent.s3.amazonaws.com/darwin/arm64/latest/amazon-cloudwatch-agent.pkg.sig https://amazoncloudwatch-agent-*region*.s3.*region*.amazonaws.com/darwin/arm64/latest/amazon-cloudwatch-agent.pkg.sig

コマンドラインを使用して CloudWatch エージェントパッケージをインストールするには

1. CloudWatch エージェントをダウンロードします。

Linux サーバーで、次のように入力します。*download-link* の場合、前の表の適切なダウンロードリンクを使用します。

```
wget download-link
```

Windows Server を実行しているサーバーで、次のファイルをダウンロードします。

```
https://amazoncloudwatch-agent.s3.amazonaws.com/windows/amd64/latest/amazon-cloudwatch-agent.msi
```

2. パッケージをダウンロードしたら、オプションでパッケージの署名を確認できます。詳細については、「CloudWatch エージェントパッケージの署名の検証」を参照してください。
3. パッケージをインストールします。Linux サーバーに RPM パッケージをダウンロードした場合は、パッケージがあるディレクトリに移動し、以下を入力します。

```
sudo rpm -U ./amazon-cloudwatch-agent.rpm
```

Linux サーバーに DEB パッケージをダウンロードした場合は、パッケージがあるディレクトリに移動し、以下を入力します。

```
sudo dpkg -i -E ./amazon-cloudwatch-agent.deb
```

Windows Server を実行するサーバーに MSI パッケージをダウンロードした場合は、パッケージがあるディレクトリに移動し、以下を入力します。

```
msiexec /i amazon-cloudwatch-agent.msi
```

このコマンドは PowerShell 内からも使用できます。MSI コマンドオプションの詳細については、Microsoft Windows のドキュメントで「Command-Line Options」を参照してください。

macOS サーバーに PKG パッケージをダウンロードした場合は、パッケージがあるディレクトリに移動し、以下を入力します。

```
sudo installer -pkg ./amazon-cloudwatch-agent.pkg -target /
```

エージェントの設定ファイルを作成して変更する

CloudWatch エージェントをダウンロードしたら、サーバーでエージェントを開始する前に設定ファイルを作成する必要があります。詳細については、「CloudWatch エージェント設定ファイルを作成する」を参照してください。

CloudWatch エージェントで使用する IAM ロールとユーザーを作成する

AWS リソースにアクセスするには、アクセス権限が必要です。IAM ロール、IAM ユーザー、またはその両方を作成して、CloudWatch エージェントが CloudWatch にメトリクスを書き込むために必要なアクセス許可を付与します。Amazon EC2 インスタンスでエージェントを使用する場合は、IAM ロールを作成する必要があります。オンプレミスサーバーでエージェントを使用する場合は、IAM ユーザーを作成する必要があります。

> **Note**
>
> お客様にこれらのポリシーを自分で作成するよう求める代わりに、Amazon が作成した新しい `CloudWatchAgentServerPolicy` および `CloudWatchAgentAdminPolicy` ポリシーを使用して、最近以下の手順を変更しました。Parameter Store へのファイル書き込みおよびファイルのダウンロードについては、Amazon によって作成されたポリシーで、`AmazonCloudWatch-` で始まる名前を持つファイルのみサポートします。ファイル名が `AmazonCloudWatch-` で始まらない CloudWatch エージェント設定ファイルがある場合、これらのポリシーを使用してファイルを Parameter Store に書き込んだり、Parameter Store からダウンロードしたりすることはできません。

Amazon EC2 インスタンスで CloudWatch エージェントを実行する場合は、次の手順を使用して必要な IAM ロールを作成します。このロールでは、インスタンスから情報を読み取り、CloudWatch に書き込むアクセス権限が提供されます。

EC2 インスタンスで CloudWatch エージェントを実行するのに必要な IAM ロールを作成するには

1. AWS Management Console にサインインして、IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
2. 左側のナビゲーションペインで、[ロール]、[ロールの作成] の順に選択します。
3. [信頼されたエンティティタイプ] で、[AWS のサービス] が選択されていることを確認します。
4. [Use case] (ユースケース) の [Common use cases] (一般的なユースケース) で、[EC2] を選択します。

5. [Next] を選択します。
6. ポリシーのリストで、[CloudWatchAgentServerPolicy] の横にあるチェックボックスを選択します。必要に応じて、検索ボックスを使用してポリシーを見つけます。
7. (オプション) エージェントが X-Ray にトレースを送信する場合はまた、AWSXRayDaemonWriteAccess ポリシーをロールに付与する必要があります。これを実行するには、リスト内でそのポリシーを見つけて、その横にあるチェックボックスを選択します。
8. [Next] を選択します。
9. [ロール名] に、ロールの名前 (例: *CloudWatchAgentServerRole*) を入力します。必要に応じて説明を入力します。続いて、[Create role] を選択します。

   これでロールが作成されました。
10. (オプション) エージェントが CloudWatch Logs にログを送信する予定であり、エージェントがこれらのロググループの保持ポリシーを設定できるようにする場合は、ロールに `logs:PutRetentionPolicy` 許可を追加する必要があります。詳細については、「CloudWatch エージェントによるログの保持ポリシーの設定を許可」を参照してください。

オンプレミスサーバーで CloudWatch エージェントを実行する場合は、次のステップを使用して、必要な IAM ユーザーを作成します。

> **Warning**
>
> このシナリオでは、プログラムによるアクセスと長期的な認証情報を持つ IAM ユーザーが必要です。これはセキュリティ上のリスクをもたらします。このリスクを軽減するために、これらのユーザーにはタスクの実行に必要な権限のみを付与し、不要になったユーザーを削除することをお勧めします。アクセスキーは、必要に応じて更新できます。詳細については、「IAM ユーザーガイド」の「アクセスキーの更新」を参照してください。

CloudWatch エージェントをオンプレミスサーバーで実行するために必要な IAM ユーザーを作成するには

1. AWS Management Console にサインインして、IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
2. 左側のナビゲーションペインで [Users] (ユーザー)、[Add users] (ユーザーの追加) の順に選択します。
3. 新しいユーザーのユーザー名を入力します。

4. [Access key - Programmatic access] (アクセスキー - プログラムによるアクセス)、[Next: Permissions] (次へ: 許可) の順に選択します。
5. [Attach existing policies directly (既存のポリシーを直接アタッチする)] を選択します。
6. ポリシーのリストで、[CloudWatchAgentServerPolicy] の横にあるチェックボックスを選択します。必要に応じて、検索ボックスを使用してポリシーを見つけます。
7. (オプション) エージェントが X-Ray にトレースを送信する場合は、AWSXRayDaemonWriteAccess ポリシーをロールに付与する必要もあります。これを実行するには、リスト内でそのポリシーを見つけて、その横にあるチェックボックスを選択します。
8. [Next: Tags] (次へ: タグ) を選択します。
9. オプションで、新しい IAM ユーザーのタグを作成し、[Next:Review] (次へ: 確認) を選択します。
10. 適切なポリシーがリストされていることを確認し、[Create user] (ユーザーを作成) を選択します。
11. 新しいユーザーの名前の横にある [Show] を選択します。エージェントのイントール時に使用できるように、アクセスキーとシークレットキーをファイルにコピーします。[閉じる] を選択します。

CloudWatch エージェントによるログの保持ポリシーの設定を許可

CloudWatch エージェントを構成して、ログイベントの送信先となるロググループ用の保持ポリシーを設定できます。これを実行する場合は、エージェントが使用する IAM ロールまたはユーザに `logs:PutRetentionPolicy` を付与する必要があります。エージェントは IAM ロールを使用して Amazon EC2 インスタンスで実行し、オンプレミスのサーバーに IAM ユーザーを使用します。

CloudWatch エージェントの IAM ロールにログの保持ポリシーを設定するための許可を付与するには

1. AWS Management Console にサインインして、IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
2. 左のナビゲーションペインで、[ロール] を選択します。
3. 検索ボックスで、CloudWatch エージェントの IAM ロールの名前の先頭を入力します。この名前は、ロールの作成時に選択されました。`CloudWatchAgentServerRole` という名前が付けられる場合があります。

   ロールが表示されたら、ロールの名前を選択します。

4. [Permissions] (許可) タブで、[Add permissions] (許可の追加)、[Create inline policy] (インラインポリシーの作成) をクリックします。
5. [JSON] タブを選択し、次のポリシーをボックスにコピーして、ボックスのデフォルトの JSON を置き換えます。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "logs:PutRetentionPolicy",
      "Resource": "*"
    }
  ]
}
```

6. [ポリシーの確認] を選択します。
7. [Name] (名前) で、**CloudWatchAgentPutLogsRetention** などと入力し、[Create policy] (ポリシーを作成) を選択します。

CloudWatch エージェントの IAM ユーザーにログ保持ポリシーを設定するための許可を付与するには

1. AWS Management Console にサインインして、IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
2. 左のナビゲーションペインで、[ユーザー] を選択します。
3. 検索ボックスで、CloudWatch エージェントの IAM ユーザーの名前の先頭を入力します。この名前は、ユーザーの作成時に選択されました。

   ユーザーが表示されたら、ユーザーの名前を選択します。
4. [アクセス許可] タブで [インラインポリシーの追加] を選択します。
5. [JSON] タブを選択し、次のポリシーをボックスにコピーして、ボックスのデフォルトの JSON を置き換えます。

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

```
      "Effect": "Allow",
      "Action": "logs:PutRetentionPolicy",
      "Resource": "*"
    }
  ]
}
```

6. [ポリシーの確認] を選択します。
7. [Name] (名前) で、**CloudWatchAgentPutLogsRetention** などと入力し、[Create policy] (ポリシーを作成) を選択します。

サーバーでの CloudWatch エージェントのインストールおよび実行

必要なエージェント設定ファイルを作成し、IAM ロールまたは IAM ユーザーを作成したら、次のステップに従い、作成した設定を使用してサーバーにエージェントをインストールして実行します。まず、エージェントを実行するサーバーに IAM ロールまたは IAM ユーザーをアタッチします。次に、そのサーバーにエージェントパッケージをダウンロードし、作成したエージェントの設定を使用して起動します。

S3 ダウンロードリンクを使用して CloudWatch エージェントパッケージをダウンロードする

> **Note**
>
> CloudWatch エージェントをダウンロードするには、接続に TLS 1.2 以降を使用する必要があります。

エージェントはその実行先となる各サーバーにインストールする必要があります。

Amazon Linux AMI

CloudWatch エージェントは、Amazon Linux 2023 および Amazon Linux 2 でパッケージとして利用できます。このオペレーティングシステムを使用している場合は、以下のコマンドを入力してパッケージをインストールできます。また、インスタンスにアタッチされた IAM ロールに CloudWatchAgentServerPolicy がアタッチされていることを確認する必要もあります。詳細については、「Amazon EC2 インスタンスの CloudWatch エージェントで使用する IAM ロールを作成する」を参照してください。

```
sudo yum install amazon-cloudwatch-agent
```

すべてのオペレーティングシステム

サポートされているすべてのオペレーティングシステムで、以下の手順で説明するように、コマンドラインと Amazon S3 ダウンロードリンクを使用して、CloudWatch エージェントをダウンロードしてインストールできます。

ダウンロードリンクごとに、全般的なリンクと各リージョンのリンクがあります。例えば、Amazon Linux 2023、Amazon Linux 2、x86-64 アーキテクチャの場合、有効なダウンロードリンクは次の 3 つです。

- `https://amazoncloudwatch-agent.s3.amazonaws.com/amazon_linux/amd64/latest/amazon-cloudwatch-agent.rpm`
- `https://amazoncloudwatch-agent-us-east-1.s3.us-east-1.amazonaws.com/amazon_linux/amd64/latest/amazon-cloudwatch-agent.rpm`
- `https://amazoncloudwatch-agent-eu-central-1.s3.eu-central-1.amazonaws.com/amazon_linux/amd64/latest/amazon-cloudwatch-agent.rpm`

アーキテクチャ	プラットフォーム	ダウンロードリンク	署名ファイルリンク
x86-64	Amazon Linux 2023 および Amazon Linux 2	https://amazoncloudwatch-agent.s3.amazonaws.com/amazon_linux/amd64/latest/amazon-cloudwatch-agent.rpm https://amazoncloudwatch-agent-*region*.s3.*region*.amazonaws.com/amazon_linux/amd64/latest/amazon-cloudwatch-agent.rpm	https://amazoncloudwatch-agent.s3.amazonaws.com/amazon_linux/amd64/latest/amazon-cloudwatch-agent.rpm.sig https://amazoncloudwatch-agent-*region*.s3.*region*.amazonaws.com/amazon_linux/amd64/latest/amazon-cloudwatch-agent.rpm.sig
x86-64	CentOS	https://amazoncloudwatch-agent.s3.amazonaws.com/centos/amd64/latest/amazon-cloudwatch-agent.rpm	https://amazoncloudwatch-agent.s3.amazonaws.com/centos/amd64/latest/amazon-cloudwatch-agent.rpm.sig

アーキテクチャ	プラットフォーム	ダウンロードリンク	署名ファイルリンク
		https://amazoncloudwatch-agent-*region*.s3.*region*.amazonaws.com/centos/amd64/latest/amazon-cloudwatch-agent.rpm	https://amazoncloudwatch-agent-*region*.s3.*region*.amazonaws.com/centos/amd64/latest/amazon-cloudwatch-agent.rpm.sig
x86-64	Redhat	https://amazoncloudwatch-agent.s3.amazonaws.com/redhat/amd64/latest/amazon-cloudwatch-agent.rpm https://amazoncloudwatch-agent-*region*.s3.*region*.amazonaws.com/redhat/amd64/latest/amazon-cloudwatch-agent.rpm	https://amazoncloudwatch-agent.s3.amazonaws.com/redhat/amd64/latest/amazon-cloudwatch-agent.rpm.sig https://amazoncloudwatch-agent-*region*.s3.*region*.amazonaws.com/redhat/amd64/latest/amazon-cloudwatch-agent.rpm.sig
x86-64	SUSE	https://amazoncloudwatch-agent.s3.amazonaws.com/suse/amd64/latest/amazon-cloudwatch-agent.rpm https://amazoncloudwatch-agent-*region*.s3.*region*.amazonaws.com/suse/amd64/latest/amazon-cloudwatch-agent.rpm	https://amazoncloudwatch-agent.s3.amazonaws.com/suse/amd64/latest/amazon-cloudwatch-agent.rpm.sig https://amazoncloudwatch-agent-*region*.s3.*region*.amazonaws.com/suse/amd64/latest/amazon-cloudwatch-agent.rpm.sig

アーキテクチャ	プラットフォーム	ダウンロードリンク	署名ファイルリンク
x86-64	Debian	https://amazoncloudwatch-agent.s3.amazonaws.com/debian/amd64/latest/amazon-cloudwatch-agent.deb https://amazoncloudwatch-agent-*region*.s3.*region*.amazonaws.com/debian/amd64/latest/amazon-cloudwatch-agent.deb	https://amazoncloudwatch-agent.s3.amazonaws.com/debian/amd64/latest/amazon-cloudwatch-agent.deb.sig https://amazoncloudwatch-agent-*region*.s3.*region*.amazonaws.com/debian/amd64/latest/amazon-cloudwatch-agent.deb.sig
x86-64	Ubuntu	https://amazoncloudwatch-agent.s3.amazonaws.com/ubuntu/amd64/latest/amazon-cloudwatch-agent.deb https://amazoncloudwatch-agent-*region*.s3.*region*.amazonaws.com/ubuntu/amd64/latest/amazon-cloudwatch-agent.deb	https://amazoncloudwatch-agent.s3.amazonaws.com/ubuntu/amd64/latest/amazon-cloudwatch-agent.deb.sig https://amazoncloudwatch-agent-*region*.s3.*region*.amazonaws.com/ubuntu/amd64/latest/amazon-cloudwatch-agent.deb.sig
x86-64	Oracle	https://amazoncloudwatch-agent.s3.amazonaws.com/oracle_linux/amd64/latest/amazon-cloudwatch-agent.rpm https://amazoncloudwatch-agent-*region*.s3.*region*.amazonaws.com/oracle_linux/amd64/latest/amazon-cloudwatch-agent.rpm	https://amazoncloudwatch-agent.s3.amazonaws.com/oracle_linux/amd64/latest/amazon-cloudwatch-agent.rpm.sig https://amazoncloudwatch-agent-*region*.s3.*region*.amazonaws.com/oracle_linux/amd64/latest/amazon-cloudwatch-agent.rpm.sig

アーキテクチャ	プラットフォーム	ダウンロードリンク	署名ファイルリンク
x86-64	macOS	https://amazoncloudwatch-agent.s3.amazonaws.com/darwin/amd64/latest/amazon-cloudwatch-agent.pkg https://amazoncloudwatch-agent-*region*.s3.*region*.amazonaws.com/darwin/amd64/latest/amazon-cloudwatch-agent.pkg	https://amazoncloudwatch-agent.s3.amazonaws.com/darwin/amd64/latest/amazon-cloudwatch-agent.pkg.sig https://amazoncloudwatch-agent-*region*.s3.*region*.amazonaws.com/darwin/amd64/latest/amazon-cloudwatch-agent.pkg.sig
x86-64	Windows	https://amazoncloudwatch-agent.s3.amazonaws.com/windows/amd64/latest/amazon-cloudwatch-agent.msi https://amazoncloudwatch-agent-*region*.s3.*region*.amazonaws.com/windows/amd64/latest/amazon-cloudwatch-agent.msi	https://amazoncloudwatch-agent.s3.amazonaws.com/windows/amd64/latest/amazon-cloudwatch-agent.msi.sig https://amazoncloudwatch-agent-*region*.s3.*region*.amazonaws.com/windows/amd64/latest/amazon-cloudwatch-agent.msi.sig
ARM64	Amazon Linux 2023 および Amazon Linux 2	https://amazoncloudwatch-agent.s3.amazonaws.com/amazon_linux/arm64/latest/amazon-cloudwatch-agent.rpm https://amazoncloudwatch-agent-*region*.s3.*region*.amazonaws.com/amazon_linux/arm64/latest/amazon-cloudwatch-agent.rpm	https://amazoncloudwatch-agent.s3.amazonaws.com/amazon_linux/arm64/latest/amazon-cloudwatch-agent.rpm.sig https://amazoncloudwatch-agent-*region*.s3.*region*.amazonaws.com/amazon_linux/arm64/latest/amazon-cloudwatch-agent.rpm.sig

アーキテクチャ	プラットフォーム	ダウンロードリンク	署名ファイルリンク
ARM64	Red Hat	https://amazoncloudwatch-agent.s3.amazonaws.com/redhat/arm64/latest/amazon-cloudwatch-agent.rpm https://amazoncloudwatch-agent-*region*.s3.*region*.amazonaws.com/redhat/arm64/latest/amazon-cloudwatch-agent.rpm	https://amazoncloudwatch-agent.s3.amazonaws.com/redhat/arm64/latest/amazon-cloudwatch-agent.rpm.sig https://amazoncloudwatch-agent-*region*.s3.*region*.amazonaws.com/redhat/arm64/latest/amazon-cloudwatch-agent.rpm.sig
ARM64	Ubuntu	https://amazoncloudwatch-agent.s3.amazonaws.com/ubuntu/arm64/latest/amazon-cloudwatch-agent.deb https://amazoncloudwatch-agent-*region*.s3.*region*.amazonaws.com/ubuntu/arm64/latest/amazon-cloudwatch-agent.deb	https://amazoncloudwatch-agent.s3.amazonaws.com/ubuntu/arm64/latest/amazon-cloudwatch-agent.deb.sig https://amazoncloudwatch-agent-*region*.s3.*region*.amazonaws.com/ubuntu/arm64/latest/amazon-cloudwatch-agent.deb.sig
ARM64	SUSE	https://amazoncloudwatch-agent.s3.amazonaws.com/suse/arm64/latest/amazon-cloudwatch-agent.rpm https://amazoncloudwatch-agent-*region*.s3.*region*.amazonaws.com/suse/arm64/latest/amazon-cloudwatch-agent.rpm	https://amazoncloudwatch-agent.s3.amazonaws.com/suse/arm64/latest/amazon-cloudwatch-agent.rpm.sig https://amazoncloudwatch-agent-*region*.s3.*region*.amazonaws.com/suse/arm64/latest/amazon-cloudwatch-agent.rpm.sig

コマンドラインを使用して CloudWatch エージェントを Amazon EC2 インスタンスにインストールするには

1. CloudWatch エージェントをダウンロードします。Linux サーバーで、次のように入力します。*download-link* の場合、前の表の適切なダウンロードリンクを使用します。

   ```
   wget download-link
   ```

   Windows Server を実行しているサーバーの場合、次のファイルをダウンロードします。

   ```
   https://amazoncloudwatch-agent.s3.amazonaws.com/windows/amd64/latest/amazon-cloudwatch-agent.msi
   ```

2. パッケージをダウンロードしたら、オプションでパッケージの署名を確認できます。詳細については、「CloudWatch エージェントパッケージの署名の検証」を参照してください。
3. パッケージをインストールします。Linux サーバーに RPM パッケージをダウンロードした場合は、パッケージがあるディレクトリに移動し、以下を入力します。

   ```
   sudo rpm -U ./amazon-cloudwatch-agent.rpm
   ```

   Linux サーバーに DEB パッケージをダウンロードした場合は、パッケージがあるディレクトリに移動し、以下を入力します。

   ```
   sudo dpkg -i -E ./amazon-cloudwatch-agent.deb
   ```

   Windows Server を実行するサーバーに MSI パッケージをダウンロードした場合は、パッケージがあるディレクトリに移動し、以下を入力します。

   ```
   msiexec /i amazon-cloudwatch-agent.msi
   ```

   このコマンドは PowerShell 内からも使用できます。MSI コマンドオプションの詳細については、Microsoft Windows のドキュメントで「Command-Line Options」を参照してください。

(EC2 インスタンスへのインストール) IAM ロールのアタッチ

CloudWatch エージェントで、インスタンスのデータを送信できるようにするには、IAM ロールをインスタンスにアタッチする必要があります。アタッチするロールは CloudWatchAgentServerRole で

す。このロールは事前に作成しておく必要があります。詳細については、「CloudWatch エージェントで使用する IAM ロールとユーザーを作成する」を参照してください。

インスタンスに IAM ロールをアタッチする方法の詳細については、Windows インスタンス用 Amazon EC2 ユーザーガイドの「IAM ロールをインスタンスにアタッチする」を参照してください。

(オンプレミスサーバーへのインストール) IAM 認証情報と AWS リージョンを指定する

CloudWatch エージェントで、オンプレミスサーバーのデータを送信できるようにするには、先ほど作成した IAM ユーザーのアクセスキーおよびシークレットキーを指定する必要があります。このユーザーの作成方法については、「CloudWatch エージェントで使用する IAM ロールとユーザーを作成する」を参照してください。

また、メトリクスを送信する AWS リージョンを指定する必要があります。そのためには、次の例に示されているように、region 設定ファイルの [AmazonCloudWatchAgent] セクションの AWS フィールドを使用します。

```
[profile AmazonCloudWatchAgent]
region = us-west-1
```

aws configure コマンドを使用して CloudWatch エージェントの名前付きプロファイルを作成する例を以下に示します。この例では、AmazonCloudWatchAgent のデフォルトプロファイル名を使用しているものとします。

CloudWatch エージェントの AmazonCloudWatchAgent プロファイルを作成するには

1. まだの場合は、AWS Command Line Interface をサーバーにインストールしてください。詳細については、「AWS CLI のインストール」を参照してください。
2. Linux サーバーでは、以下のコマンドを入力し、プロンプトに従います。

   ```
   sudo aws configure --profile AmazonCloudWatchAgent
   ```

   Windows Server では、管理者として PowerShell を開き、以下のコマンドを入力して、プロンプトに従います。

   ```
   aws configure --profile AmazonCloudWatchAgent
   ```

インターネットアクセスを確認する

CloudWatch または CloudWatch Logs にデータを送信するには、Amazon EC2 インスタンスにアウトバウンドインターネットアクセスが必要です。インターネットアクセスの詳しい設定方法については、Amazon VPC ユーザーガイドの「インターネットゲートウェイ」を参照してください。

プロキシで設定するエンドポイントとポートは、次のとおりです。

- エージェントを使用してメトリクスを収集する場合は、適切なリージョンの CloudWatch エンドポイントを許可リストに追加する必要があります。これらのエンドポイントは、「Amazon CloudWatch エンドポイントとクォータ」に記載されています。
- エージェントを使用してログを収集する場合は、適切なリージョンの CloudWatch Logs エンドポイントを許可リストに追加する必要があります。これらのエンドポイントは、「Amazon CloudWatch Logs エンドポイントとクォータ」に記載されています。
- Systems Manager を使用してエージェントをインストールするか、Parameter Store を使用して設定ファイルを保存する場合は、適切なリージョンの Systems Manager エンドポイントを許可リストに追加する必要があります。これらのエンドポイントは、「AWS Systems Manager エンドポイントとクォータ」に記載されています。

(オプション) プロキシ情報またはリージョン情報の一般的な設定を変更する

CloudWatch エージェントには、`common-config.toml` と呼ばれる設定ファイルが含まれます。必要に応じてこのファイルを使用して、プロキシおよびリージョン情報を指定できます。

Linux を実行しているサーバーでは、このファイルは `/opt/aws/amazon-cloudwatch-agent/etc` ディレクトリにあります。Windows Server を実行しているサーバーでは、このファイルは `C:\ProgramData\Amazon\AmazonCloudWatchAgent` ディレクトリにあります。

> **Note**
>
> CloudWatch エージェントをオンプレミスモードで実行する場合は、`common-config.toml` ファイルを使用して共有設定および認証情報を提供することをお勧めします。また、Amazon EC2 上で実行していて、既存の共有認証情報プロファイルおよびファイルを再利用する場合にも役立ちます。`common-config.toml` を使用して有効にすると、有効期限が切れた後に共有認証情報ファイルが更新された認証情報でローテーションされた場合、再起動せずに新しい認証情報がエージェントによって自動的に取得されるという利点もあります。

デフォルトの common-config.toml は次のとおりです。

```
# This common-config is used to configure items used for both ssm and cloudwatch access


## Configuration for shared credential.
## Default credential strategy will be used if it is absent here:
##            Instance role is used for EC2 case by default.
##            AmazonCloudWatchAgent profile is used for the on-premises case by
 default.
# [credentials]
#    shared_credential_profile = "{profile_name}"
#    shared_credential_file= "{file_name}"

## Configuration for proxy.
## System-wide environment-variable will be read if it is absent here.
## i.e. HTTP_PROXY/http_proxy; HTTPS_PROXY/https_proxy; NO_PROXY/no_proxy
## Note: system-wide environment-variable is not accessible when using ssm run-command.
## Absent in both here and environment-variable means no proxy will be used.
# [proxy]
#    http_proxy = "{http_url}"
#    https_proxy = "{https_url}"
#    no_proxy = "{domain}"
```

初期状態では、すべての行がコメントアウトされています。認証情報プロファイルやプロキシ設定を設定するには、その行から # を削除し、値を指定します。このファイルを手動で編集するか、Systems Manager の RunShellScript Run Command を使用できます。

- shared_credential_profile – オンプレミスサーバーの場合、このコード行では、データを CloudWatch に送信するために使用する IAM ユーザー認証情報を指定します。このコード行をコメントアウトした場合、AmazonCloudWatchAgent が使用されます。このプロファイルの作成の詳細については、「(オンプレミスサーバーへのインストール) IAM 認証情報と AWS リージョンを指定する」を参照してください。

  EC2 インスタンスでは、このコード行を使用して、CloudWatch エージェントでこのインスタンスのデータを別の AWS リージョンの CloudWatch に送信します。そのためには、送信先のリージョンの名前を指定する region フィールドを含む名前付きプロファイルを指定します。

  shared_credential_profile を指定する場合は、[credentials] 行の先頭から # を削除する必要もあります。

- shared_credential_file – エージェントでデフォルトのパス以外のパスにあるファイルの認証情報を検索するには、ここで完全なパスとファイル名を指定します。デフォルトのパスは、/root/.aws (Linux) および C:\\Users\\Administrator\\.aws (Windows Server) です。

  以下の最初の例は、Linux サーバーで有効な shared_credential_file 行の構文です。2 番目の例は、Windows Server で有効です。Windows Server では、\ 文字をエスケープする必要があります。

  ```
  shared_credential_file= "/usr/username/credentials"
  ```

  ```
  shared_credential_file= "C:\\Documents and Settings\\username\\.aws\\credentials"
  ```

  shared_credential_file を指定する場合は、[credentials] 行の先頭から # を削除する必要もあります。

- プロキシ設定 – サーバーで HTTP プロキシまたは HTTPS プロキシを使用して AWS サービスに接続する場合は、これらのプロキシを http_proxy フィールドと https_proxy フィールドに指定します。プロキシから除外すべき URL がある場合、カンマで区切って no_proxy フィールドで指定します。

コマンドラインを使用して CloudWatch エージェントを起動する

コマンドラインを使用してサーバー上の CloudWatch エージェントを起動するには、以下のステップを実行します。

コマンドラインを使用してサーバーの CloudWatch エージェントを起動するには

1. 使用するエージェント設定ファイルを、エージェントを実行するサーバーにコピーします。設定ファイルのコピー先のパス名を書き留めます。
2. このコマンドで、-a fetch-config によりエージェントは最新バージョンの CloudWatch エージェント設定ファイルをロードし、-s がエージェントを開始します。

   以下のいずれかのコマンドを入力します。*configuration-file-path* は、エージェント設定ファイルへのパスに置き換えます。このファイルの名前は、ウィザードで作成した場合は config.json になり、手動で作成した場合は amazon-cloudwatch-agent.json になる場合があります。

   Linux を実行する EC2 インスタンスで、次のコマンドを入力します。

```
sudo /opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-ctl -a fetch-
config -m ec2 -s -c file:configuration-file-path
```

Linux を実行しているオンプレミスサーバーでは、次のように入力します。

```
sudo /opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-ctl -a fetch-
config -m onPremise -s -c file:configuration-file-path
```

Windows Server を実行する EC2 インスタンスでは、PowerShell コンソールから次のように入力します。

```
& "C:\Program Files\Amazon\AmazonCloudWatchAgent\amazon-cloudwatch-agent-ctl.ps1" -
a fetch-config -m ec2 -s -c file:configuration-file-path
```

Windows Server を実行するオンプレミスサーバーでは、PowerShell コンソールから次のように入力します。

```
& "C:\Program Files\Amazon\AmazonCloudWatchAgent\amazon-cloudwatch-agent-ctl.ps1" -
a fetch-config -m onPremise -s -c file:configuration-file-path
```

AWS Systems Manager を使用した CloudWatch エージェントのインストール

以下のトピックに従い、 を使用して CloudWatch エージェントをインストールおよび実行します AWS Systems Manager

トピック

- CloudWatch エージェントで使用する IAM ロールとユーザーを作成する
- CloudWatch エージェントをダウンロードして設定する
- エージェント設定を使用した EC2 インスタンスへの CloudWatch エージェントのインストール
- オンプレミスサーバーへの CloudWatch エージェントのインストール

CloudWatch エージェントで使用する IAM ロールとユーザーを作成する

AWS リソースにアクセスするには、アクセス権限が必要です。CloudWatch エージェントが CloudWatch にメトリクスを書き込み、CloudWatch エージェントが Amazon EC2 および と通信するために必要な許可を含む IAM ロールとユーザーを作成できますAWS Systems Manager Amazon EC2 インスタンスでは IAM ロールを使用し、オンプレミスのサーバーでは IAM ユーザーを使用します。

1 つのロールまたはユーザーを使用することで、CloudWatch エージェントをサーバーにインストールし、メトリクスを CloudWatch に送信できます。CloudWatch エージェントの設定を Systems Manager Parameter Store に保存するには、もう 1 つのロールまたはユーザーが必要です。Parameter Store により、複数のサーバーで 1 つの CloudWatch エージェント設定を使用できます。

Parameter Store に書き込む機能は、広範で強力なアクセス許可です。このアクセス許可は、必要な場合にのみ使用します。デプロイの複数のインスタンスにアタッチしないでください。CloudWatch エージェントの設定を Parameter Store に保存する場合は、次のことを推奨します。

- この設定を実行するインスタンスを 1 つセットアップします。
- このインスタンスでのみ Parameter Store に書き込むには、アクセス許可を持つ IAM ロールを使用します。
- CloudWatch エージェント設定ファイルの使用中および保存中のみ、Parameter Store への書き込みアクセス許可を持つ IAM ロールを使用します。

> **Note**
>
> お客様にこれらのポリシーを自分で作成するよう求める代わりに、Amazon が作成した新しい `CloudWatchAgentServerPolicy` および `CloudWatchAgentAdminPolicy` ポリシーを使用して、最近以下の手順を変更しました。これらのポリシーを使用してエージェント設定ファイルを Parameter Store に書き込み、次に Parameter Store からダウンロードするには、エージェント設定ファイルの名前を `AmazonCloudWatch-` で始める必要があります。ファイル名が `AmazonCloudWatch-` で始まらない CloudWatch エージェント設定ファイルがある場合、これらのポリシーを使用してファイルを Parameter Store に書き込んだり、Parameter Store からファイルをダウンロードしたりすることはできません。

Amazon EC2 インスタンスの CloudWatch エージェントで使用する IAM ロールを作成する

最初の手順では、CloudWatch エージェントで実行される各 Amazon EC2 インスタンスにアタッチする必要がある IAM ロールが作成されます。このロールでは、インスタンスから情報を読み取り、CloudWatch に書き込むアクセス権限が提供されます。

2 番目の手順で作成する IAM ロールは、CloudWatch エージェント設定ファイルの作成に使用する Amazon EC2 インスタンスにアタッチします。この手順は、このファイルを Systems Manager Parameter Store に保存して他のサーバーで使用できるようにする場合に必要です。このロールでは、Parameter Store に書き込むアクセス権限と、インスタンスから情報を読み取って CloudWatch に書き込むアクセス権限が提供されます。このロールには、CloudWatch エージェントを実行するだけでなく、Parameter Store に書き込むのに十分なアクセス権限が含まれます。

> **Note**
>
> Parameter Store では、標準とアドバンストの階層のパラメータをサポートしています。これらのパラメータ層は、CloudWatch エージェントの定義済みメトリクスセットで使用可能な詳細のベーシックレベル、標準レベル、アドバンストレベルとは関係ありません。

各サーバーが CloudWatch エージェントを実行するのに必要な IAM ロールを作成するには

1. AWS Management Console にサインインして、IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
2. ナビゲーションペインで [ロール] を選択した後、[ロールの作成] を選択します。
3. [Select type of trusted entity] (信頼されたエンティティの種類を選択) の下で、[AWS service] (AWS サービス) を選択します。
4. [Common use cases (一般的ユースケース)] で [EC2 (EC2)] を選択し、[Next: Permissions (次へ: アクセス許可)] を選択します。
5. ポリシーのリストで、[CloudWatchAgentServerPolicy] の横にあるチェックボックスを選択します。必要に応じて、検索ボックスを使用してポリシーを見つけます。
6. Systems Manager を使用して CloudWatch エージェントをインストールまたは設定するには、[AmazonSSMManagedInstanceCore] の横にあるボックスを選択します。この AWS 管理ポリシーにより、インスタンスは Systems Manager サービスコア機能を使用できます。必要に応じて、検索ボックスを使用してポリシーを見つけます。コマンドラインのみ使用してエージェントを開始および設定する場合、このポリシーは必要ありません。
7. [次へ: タグ] を選択します。

8. (オプション) 1 つ以上のタグ/値ペアを追加して、このロールのアクセスを整理、追跡、または制御し、[次へ: 確認] を選択します。
9. [ロール名] に、新しいロールの名前を入力 (**CloudWatchAgentServerRole** など) するか、希望する別の名前を入力します。
10. (オプション) [Role description] (ロールの説明) に、説明を入力します。
11. [AmazonSSMManagedInstanceCore] およびオプションで [CloudWatchAgentServerPolicy] が [ポリシー] の横に表示されることを確認します。
12. [ロールの作成] を選択します。

    これでロールが作成されました。

次の手順では、IAM ロールを作成します。このロールは、Parameter Store に書き込むこともできます。このロールを使用してエージェント設定ファイルを Parameter Store に保存し、他のサーバーがそのファイルを取得するようにできます。

Parameter Store への書き込みアクセス許可により、広範なアクセスが提供されます。このロールは、すべてのサーバーにアタッチせずに、管理者のみ使用できるようにします。エージェント設定ファイルを作成し、Parameter Store にコピーしたら、インスタンスからこのロールをデタッチし、代わりに `CloudWatchAgentServerRole` を使用してください。

管理者が Parameter Store に書き込むための IAM ロールを作成するには

1. AWS Management Console にサインインして、IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
2. ナビゲーションペインで [ロール] を選択した後、[ロールの作成] を選択します。
3. [Select type of trusted entity] (信頼されたエンティティの種類を選択) の下で、[AWS service] (AWS サービス) を選択します。
4. [このロールを使用するサービスを選択] のすぐ下で、[EC2]、[次へ: アクセス許可)] を選択します。
5. ポリシーのリストで、[CloudWatchAgentAdminPolicy] の横にあるチェックボックスを選択します。必要に応じて、検索ボックスを使用してポリシーを見つけます。
6. Systems Manager を使用して CloudWatch エージェントをインストールまたは設定するには、[AmazonSSMManagedInstanceCore] の横にあるボックスを選択します。この AWS 管理ポリシーにより、インスタンスは Systems Manager サービスコア機能を使用できます。必要に応じて、検索ボックスを使用してポリシーを見つけます。コマンドラインのみ使用してエージェントを開始および設定する場合、このポリシーは必要ありません。

7. [次へ: タグ] を選択します。
8. (オプション) 1 つ以上のタグ/値ペアを追加して、このロールのアクセスを整理、追跡、または制御し、[次へ: 確認] を選択します。
9. [ロール名] に、新しいロールの名前を入力 (**CloudWatchAgentAdminRole** など) するか、希望する別の名前を入力します。
10. (オプション) [Role description] (ロールの説明) に、説明を入力します。
11. [CloudWatchAgentAdminPolicy] およびオプションで [AmazonSSMManagedInstanceCore] が [ポリシー] の横に表示されることを確認します。
12. [ロールの作成] を選択します。

    これでロールが作成されました。

オンプレミスサーバーで CloudWatch エージェントを使用するための IAM ユーザーを作成する

最初の手順では、CloudWatch エージェントの実行に必要な IAM ユーザーが作成されます。このユーザーは、CloudWatch にデータを送信するアクセス許可を提供します。

2 番目の手順では、CloudWatch エージェント設定ファイルを作成するときに使用できる IAM ユーザーを作成します。このファイルを Systems Manager Parameter Store に保存して、他のサーバーで使用できるようにするには、この手順を使用します。このユーザーは、Parameter Store にデータを書き込むアクセス許可に加えて、CloudWatch に書き込むアクセス許可を提供します。

> **Note**
>
> Parameter Store では、標準とアドバンストの階層のパラメータをサポートしています。これらのパラメータ層は、CloudWatch エージェントの定義済みメトリクスセットで使用可能な詳細のベーシックレベル、標準レベル、アドバンストレベルとは関係ありません。

CloudWatch エージェントが CloudWatch にデータを書き込むために必要な IAM ユーザーを作成するには

1. AWS Management Console にサインインして、IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
2. ナビゲーションペインで [ユーザー]、[ユーザーの追加] の順に選択します。
3. 新しいユーザーのユーザー名を入力します。

4. [アクセスの種類] で、[プログラムによるアクセス] を選択し、[次の手順: アクセス許可] を選択します。
5. [許可を設定] で、[Attach existing policies directly (既存のポリシーを直接アタッチする)] を選択します。
6. ポリシーのリストで、[CloudWatchAgentServerPolicy] の横にあるチェックボックスを選択します。必要に応じて、検索ボックスを使用してポリシーを見つけます。
7. Systems Manager を使用して CloudWatch エージェントをインストールまたは設定するには、[AmazonSSMManagedInstanceCore] の横にあるボックスを選択します。この AWS 管理ポリシーにより、インスタンスは Systems Manager サービスコア機能を使用できます。(必要に応じて、検索ボックスを使用してポリシーを検索します。コマンドラインのみを使用してエージェントを開始および設定する場合、このポリシーは必要ありません。)
8. [次へ: タグ] を選択します。
9. (オプション) 1 つ以上のタグ/値ペアを追加して、このロールのアクセスを整理、追跡、または制御し、[次へ: 確認] を選択します。
10. 適切なポリシーが表示されていることを確認し、[ユーザーの作成] を選択します。
11. 新しいユーザーの行で、[表示] を選択します。エージェントのイントール時に使用できるように、アクセスキーとシークレットキーをファイルにコピーします。[閉じる] を選択します。

次の手順では、IAM ユーザーを作成します。このユーザーは、Parameter Store に書き込むこともできます。他のサーバーが使用できるようにエージェント設定ファイルを Parameter Store に保存する場合は、この IAM ユーザーを使用する必要があります。この IAM ユーザーは、Parameter Store に書き込むためのアクセス許可を提供します。また、インスタンスから情報を読み取り、CloudWatch に書き込むアクセス許可も提供します。Systems Manager Parameter Store に書き込むためのアクセス許可は、広範なアクセスを提供します。この IAM ユーザーは、すべてのサーバーにアタッチせずに、管理者のみ使用できるようにします。この IAM ユーザーは、エージェント設定ファイルを Parameter Store に保存するときにのみ使用してください。

Parameter Store のファイル設定を保存し、CloudWatch に情報を送信するために必要な IAM ユーザーを作成するには

1. AWS Management Console にサインインして、IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
2. ナビゲーションペインで [ユーザー]、[ユーザーの追加] の順に選択します。
3. 新しいユーザーのユーザー名を入力します。

4. [アクセスの種類] で、[プログラムによるアクセス] を選択し、[次の手順: アクセス許可] を選択します。
5. [許可を設定] で、[Attach existing policies directly (既存のポリシーを直接アタッチする)] を選択します。
6. ポリシーのリストで、[CloudWatchAgentAdminPolicy] の横にあるチェックボックスを選択します。必要に応じて、検索ボックスを使用してポリシーを見つけます。
7. Systems Manager を使用して CloudWatch エージェントをインストールまたは設定するには、[AmazonSSMManagedInstanceCore] の横にあるチェックボックスをオンにします。この AWS 管理ポリシーにより、インスタンスは Systems Manager サービスコア機能を使用できます。(必要に応じて、検索ボックスを使用してポリシーを検索します。コマンドラインのみを使用してエージェントを開始および設定する場合、このポリシーは必要ありません。)
8. [次へ: タグ] を選択します。
9. (オプション) 1 つ以上のタグ/値ペアを追加して、このロールのアクセスを整理、追跡、または制御し、[次へ: 確認] を選択します。
10. 適切なポリシーが表示されていることを確認し、[ユーザーの作成] を選択します。
11. 新しいユーザーの行で、[表示] を選択します。エージェントのイントール時に使用できるように、アクセスキーとシークレットキーをファイルにコピーします。[閉じる] を選択します。

CloudWatch エージェントをダウンロードして設定する

このセクションでは、Systems Manager を使用してエージェントをダウンロードする方法と、その後エージェント設定ファイルを作成する方法について説明します。Systems Manager を使用してエージェントをダウンロードするには、Systems Manager に合わせてインスタンスが適切に設定されていることを確認する必要があります。

SSM Agent のインストールと更新

Amazon EC2 インスタンスでは、CloudWatch エージェントを使用するにはバージョン 2.2.93.0 以降のインスタンスが実行されている必要があります。CloudWatch エージェントをインストールする前に、SSM Agent を更新するか、インスタンスにインストールしてください (まだの場合)。

Linux を実行しているインスタンスでの SSM Agent のインストールまたは更新については、AWS Systems Manager ユーザーガイドの Linux インスタンスでの SSM Agent のインストールおよび設定を参照してください。

SSM Agent のインストールまたは更新の詳細については、AWS Systems Manager ユーザーガイドの SSM Agent の使用を参照してください。

(オプション) Systems Manager の前提条件を確認します。

インターネットアクセスを確認する

CloudWatch または CloudWatch Logs にデータを送信するには、Amazon EC2 インスタンスにアウトバウンドインターネットアクセスが必要です。インターネットアクセスの詳しい設定方法については、Amazon VPC ユーザーガイドの「インターネットゲートウェイ」を参照してください。

プロキシで設定するエンドポイントとポートは、次のとおりです。

- エージェントを使用してメトリクスを収集する場合は、適切なリージョンの CloudWatch エンドポイントのリストを許可する必要があります。これらのエンドポイントは、「Amazon Web Services 全般のリファレンス」の「Amazon CloudWatch」に記載されています。
- エージェントを使用してログを収集する場合は、適切なリージョンの CloudWatch Logs エンドポイントのリストを許可する必要があります。これらのエンドポイントは、「Amazon Web Services 全般のリファレンス」の「Amazon CloudWatch Logs」に記載されています。
- Systems Manager を使用してエージェントをインストールする、または Parameter Store を使用して設定ファイルを保存する場合は、適切なリージョンの Systems Manager エンドポイントのリストを許可する必要があります。これらのエンドポイントは、「Amazon Web Services 全般のリファレンス」の「AWS Systems Manager」に記載されています。

Systems Manager を使用して CloudWatch エージェントパッケージをダウンロードするには、次の手順に従います。

Systems Manager を使用して CloudWatch エージェントをダウンロードするには

1. Systems Manager コンソール (https://console.aws.amazon.com/systems-manager/) を開きます。
2. ナビゲーションペインで、[Run Command] を選択します。

   -または-

   AWS Systems Manager のホームページが表示された場合は、スクロールダウンして [Explore Run Command] (Run Command の詳細) を選択します。
3. [Run command (コマンドの実行)] を選択します。
4. [Command document] リストで、[AWS-ConfigureAWSPackage] を選択します。
5. [ターゲット] 領域で、CloudWatch エージェントをインストールする先のインスタンスを選択します。特定のインスタンスが表示されない場合、そのインスタンスは Systems Manager で

使用するマネージドインスタンスとして設定されていない可能性があります。詳細については、AWS Systems Manager ユーザーガイドからハイブリッド環境での AWS Systems Manager のセットアップを参照してください。

6. [アクション] リストで、[インストール] を選択します。
7. [名前] フィールドに、「*AmazonCloudWatchAgent*」と入力します。
8. エージェントの最新バージョンをインストールには、[Version (バージョン)] を [latest (最新)] に設定したままにします。
9. [Run (実行)] を選択します。
10. 必要に応じて、[Targets and outputs] 領域で、インスタンス名の横のボタンを選択して [View output] を選択します。Systems Manager に、エージェントが正常にインストールされたことが表示されます。

エージェントの設定ファイルを作成して変更する

CloudWatch エージェントをダウンロードしたら、サーバーでエージェントを開始する前に設定ファイルを作成する必要があります。

エージェント設定ファイルを Systems Manager Parameter Store に保存する場合は、EC2 インスタンスを使用してParameter Store に保存する必要があります。さらに、最初にそのインスタンスに `CloudWatchAgentAdminRole` IAM ロールをアタッチする必要があります。ロールのアタッチの詳細については、Windows インスタンス用 Amazon EC2 ユーザーガイドの「IAM ロールをインスタンスにアタッチする」を参照してください。

CloudWatch エージェント設定ファイルの作成の詳細については、「CloudWatch エージェント設定ファイルを作成する」を参照してください。

エージェント設定を使用した EC2 インスタンスへの CloudWatch エージェントのインストール

CloudWatch エージェント設定を Parameter Store に保存すると、他のサーバーにエージェントをインストールするときに使用できます。

トピック

- IAM ロールをインスタンスにアタッチする
- Amazon EC2 インスタンスに CloudWatch エージェントパッケージをダウンロードする
- (オプション) CloudWatch エージェントの一般的な設定と名前付きプロファイルを変更する

- CloudWatch エージェントを起動する

IAM ロールをインスタンスにアタッチする

インスタンスで CloudWatch エージェントを実行できるようにするには CloudWatchAgentServerRole IAM ロールを EC2 インスタンスにアタッチする必要があります。このロールは、CloudWatch エージェントがインスタンス上でアクションを実行できるようにします。このロールは事前に作成しておく必要があります。詳細については、「CloudWatch エージェントで使用する IAM ロールとユーザーを作成する」を参照してください。

詳細については、Windows インスタンス用 Amazon EC2 ユーザーガイドの「IAM ロールをインスタンスにアタッチする」を参照してください。

Amazon EC2 インスタンスに CloudWatch エージェントパッケージをダウンロードする

エージェントはその実行先となる各サーバーにインストールする必要があります。CloudWatch エージェントは、Amazon Linux 2023 および Amazon Linux 2 でパッケージとして利用できます。このオペレーティングシステムを使用している場合は、以下のコマンドを入力してパッケージをインストールできます。また、インスタンスにアタッチされた IAM ロールに CloudWatchAgentServerPolicy がアタッチされていることを確認する必要もあります。詳細については、「Amazon EC2 インスタンスの CloudWatch エージェントで使用する IAM ロールを作成する」を参照してください。

```
sudo yum install amazon-cloudwatch-agent
```

サポートされているすべてのオペレーティングシステムで、Systems Manager Run Command または Amazon S3 ダウンロードリンクを使用して CloudWatch エージェントパッケージをダウンロードできます。Amazon S3 ダウンロードリンクの使用については、「CloudWatch エージェントパッケージをダウンロードする」を参照してください。

> **Note**
>
> CloudWatch エージェントをインストールまたは更新する場合、アンインストールして再インストールするオプションのみがサポートされています。インプレースの更新オプションを使用することはできません。

Systems Manager を使用して Amazon EC2 インスタンスで CloudWatch エージェントをダウンロードする

Systems Manager を使用して CloudWatch エージェントをインストールするには、Systems Manager に合わせてインスタンスが適切に設定されていることを確認する必要があります。

SSM Agent のインストールと更新

Amazon EC2 インスタンスでは、CloudWatch エージェントを使用するにはバージョン 2.2.93.0 以降のインスタンスが実行されている必要があります。CloudWatch エージェントをインストールする前に、SSM Agent を更新するか、インスタンスにインストールしてください (まだの場合)。

Linux を実行しているインスタンスでの SSM Agent のインストールまたは更新については、AWS Systems Manager ユーザーガイドの Linux インスタンスでの SSM Agent のインストールおよび設定を参照してください。

Windows Server を実行しているインスタンスでの SSM Agent のインストールまたは更新については、AWS Systems Manager ユーザーガイドの Windows インスタンスでの SSM Agent のインストールおよび設定を参照してください。

(オプション) Systems Manager の前提条件を確認します。

Systems Manager Run Command を使用して CloudWatch エージェントをインストールおよび設定する前に、インスタンスが Systems Manager の最低要件を満たしていることを確認してください。詳細については、AWS Systems Manager ユーザーガイドのAWS Systems Manager のセットアップを参照してください。

インターネットアクセスを確認する

CloudWatch または CloudWatch Logs にデータを送信するには、Amazon EC2 インスタンスにアウトバウンドインターネットアクセスが必要です。インターネットアクセスの詳しい設定方法については、Amazon VPC ユーザーガイドの「インターネットゲートウェイ」を参照してください。

CloudWatch エージェントパッケージをダウンロードする

Systems Manager Run Command では、インスタンスの設定を管理できます。Systems Manager ドキュメントを指定してパラメータを指定し、1 つ以上のインスタンスでコマンドを実行します。インスタンスの SSM Agent は、コマンドを処理し、指定されたとおりにインスタンスを設定します。

Run Command を使用して CloudWatch エージェントをダウンロードするには

1. Systems Manager コンソール (https://console.aws.amazon.com/systems-manager/) を開きます。
2. ナビゲーションペインで、[Run Command] を選択します。

   -または-

   AWS Systems Manager のホームページが表示された場合は、スクロールダウンして [Explore Run Command] (Run Command の詳細) を選択します。
3. [Run command (コマンドの実行)] を選択します。
4. [Command document] リストで、[AWS-ConfigureAWSPackage] を選択します。
5. [Targets] 領域で、CloudWatch エージェントをインストールするインスタンスを選択します。特定のインスタンスが表示されない場合、Run Command 用に設定されていない可能性があります。詳細については、AWS Systems Manager ユーザーガイドからハイブリッド環境での AWS Systems Manager のセットアップを参照してください。
6. [アクション] リストで、[インストール] を選択します。
7. [名前] ボックスに、「*AmazonCloudWatchAgent*」と入力します。
8. エージェントの最新バージョンをインストールには、[Version (バージョン)] を [latest (最新)] に設定したままにします。
9. [Run (実行)] を選択します。
10. 必要に応じて、[Targets and outputs] 領域で、インスタンス名の横のボタンを選択して [View output] を選択します。Systems Manager に、エージェントが正常にインストールされたことが表示されます。

(オプション) CloudWatch エージェントの一般的な設定と名前付きプロファイルを変更する

CloudWatch エージェントには、`common-config.toml` と呼ばれる設定ファイルが含まれます。(オプション) このファイルを使用してプロキシおよびリージョン情報を指定できます。

Linux を実行しているサーバーでは、このファイルは `/opt/aws/amazon-cloudwatch-agent/etc` ディレクトリにあります。Windows Server を実行しているサーバーでは、このファイルは `C:\ProgramData\Amazon\AmazonCloudWatchAgent` ディレクトリにあります。

デフォルトの `common-config.toml` は次のようになります。

```
# This common-config is used to configure items used for both ssm and cloudwatch access
```

```
## Configuration for shared credential.
## Default credential strategy will be used if it is absent here:
##            Instance role is used for EC2 case by default.
##            AmazonCloudWatchAgent profile is used for onPremise case by default.
# [credentials]
#    shared_credential_profile = "{profile_name}"
#    shared_credential_file= "{file_name}"

## Configuration for proxy.
## System-wide environment-variable will be read if it is absent here.
## i.e. HTTP_PROXY/http_proxy; HTTPS_PROXY/https_proxy; NO_PROXY/no_proxy
## Note: system-wide environment-variable is not accessible when using ssm run-command.
## Absent in both here and environment-variable means no proxy will be used.
# [proxy]
#    http_proxy = "{http_url}"
#    https_proxy = "{https_url}"
#    no_proxy = "{domain}"
```

初期状態では、すべての行がコメントアウトされています。認証情報プロファイルやプロキシ設定を設定するには、その行から # を削除し、値を指定します。このファイルを手動で編集するか、Systems Manager の RunShellScript Run Command を使用できます。

- shared_credential_profile – オンプレミスサーバーの場合、このコード行では、データを CloudWatch に送信するために使用する IAM ユーザー認証情報を指定します。このコード行をコメントアウトした場合、AmazonCloudWatchAgent が使用されます。

  EC2 インスタンスでは、このコード行を使用して、CloudWatch エージェントでこのインスタンスのデータを別の AWS リージョンの CloudWatch に送信します。そのためには、送信先のリージョンの名前を指定する region フィールドを含む名前付きプロファイルを指定します。

  shared_credential_profile を指定する場合は、[credentials] 行の先頭から # を削除する必要もあります。
- shared_credential_file – エージェントでデフォルトのパス以外のパスにあるファイルの認証情報を検索するには、ここで完全なパスとファイル名を指定します。デフォルトのパスは、/root/.aws (Linux) および C:\\Users\\Administrator\\.aws (Windows Server) です。

  以下の最初の例は、Linux サーバーで有効な shared_credential_file 行の構文です。2 番目の例は、Windows Server で有効です。Windows Server では、\ 文字をエスケープする必要があります。

```
shared_credential_file= "/usr/username/credentials"
```

```
shared_credential_file= "C:\\Documents and Settings\\username\\.aws\\credentials"
```

shared_credential_file を指定する場合は、[credentials] 行の先頭から # を削除する必要もあります。

- プロキシ設定 – サーバーで HTTP プロキシまたは HTTPS プロキシを使用して AWS サービスに接続する場合は、これらのプロキシを http_proxy フィールドと https_proxy フィールドに指定します。プロキシから除外すべき URL がある場合、カンマで区切って no_proxy フィールドで指定します。

CloudWatch エージェントを起動する

Systems Manager Run Command またはコマンドラインを使用してエージェントを起動できます。

Systems Manager Run Command を使用して CloudWatch エージェントを起動する

Systems Manager Run Command を使用してエージェントを起動するには、次の手順に従います。

コマンドの実行を使用して CloudWatch エージェントを起動するには

1. Systems Manager コンソール (https://console.aws.amazon.com/systems-manager/) を開きます。
2. ナビゲーションペインで、[Run Command] を選択します。

   -または-

   AWS Systems Manager のホームページが表示された場合は、スクロールダウンして [Explore Run Command] (Run Command の詳細) を選択します。
3. [Run command (コマンドの実行)] を選択します。
4. [Command document] リストで、[AmazonCloudWatch-ManageAgent] を選択します。
5. [Targets] 領域で、CloudWatch エージェントをインストールしたインスタンスを選択します。
6. [Action] リストで、[configure] を選択します。
7. [Optional Configuration Source] リストで、[ssm] を選択します。

8. [オプションの設定の場所] ボックスに、「CloudWatch エージェント設定ファイルを作成する」で説明されているように、作成して Systems Manager Parameter Store に保存したエージェント設定ファイルの Systems Manager パラメータ名を入力します。
9. これらのステップを完了した後、[Optional Restart] リストで、[yes] を選択してエージェントを開始します。
10. [Run (実行)] を選択します。
11. 必要に応じて、[Targets and outputs] 領域で、インスタンス名の横のボタンを選択して [View output] を選択します。Systems Manager に、エージェントが正常に開始されたことが表示されます。

コマンドラインを使用して Amazon EC2 インスタンスで CloudWatch エージェントを起動する

コマンドラインを使用して CloudWatch エージェントを Amazon EC2 インスタンスにインストールするには、以下の手順を実行します。

コマンドラインを使用して CloudWatch エージェントを Amazon EC2 インスタンスに起動するには

- このコマンドで、`-a fetch-config` によりエージェントは最新バージョンの CloudWatch エージェント設定ファイルをロードし、`-s` がエージェントを開始します。

  Linux および macOS: Systems Manager Parameter Store に設定ファイルを保存した場合は、次のように入力します。

  ```
  sudo /opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-ctl -a fetch-config -m ec2 -s -c ssm:configuration-parameter-store-name
  ```

  Linux および macOS: 設定ファイルをローカルコンピュータに保存した場合は、次のコマンドを入力します。*configuration-file-path* は、エージェント設定ファイルへのパスに置き換えます。このファイルの名前は、ウィザードで作成した場合は `config.json` になり、手動で作成した場合は `amazon-cloudwatch-agent.json` になる場合があります。

  ```
  sudo /opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-ctl -a fetch-config -m ec2 -s -c file:configuration-file-path
  ```

  Windows Server: エージェント設定ファイルを Systems Manager Parameter Store に保存した場合は、PowerShell コンソールから次のように入力します。

```
& "C:\Program Files\Amazon\AmazonCloudWatchAgent\amazon-cloudwatch-agent-ctl.ps1" -
a fetch-config -m ec2 -s -c ssm:configuration-parameter-store-name
```

Windows Server: エージェント設定ファイルをローカルコンピュータに保存した場合は、PowerShell コンソールから次のように入力します。

```
& "C:\Program Files\Amazon\AmazonCloudWatchAgent\amazon-cloudwatch-agent-ctl.ps1"
 -a fetch-config -m ec2 -s -c file:"C:\Program Files\Amazon\AmazonCloudWatchAgent
\config.json"
```

オンプレミスサーバーへの CloudWatch エージェントのインストール

CloudWatch エージェントをあるコンピュータにダウンロードし、必要なエージェント設定ファイルを作成した場合は、その設定ファイルを使用して他のオンプレミスサーバーにエージェントをインストールできます。

オンプレミスサーバーに CloudWatch エージェントをダウンロードする

Systems Manager Run Command または Amazon S3 ダウンロードリンクを使用して CloudWatch エージェントパッケージをダウンロードできます。Amazon S3 ダウンロードリンクの使用については、「CloudWatch エージェントパッケージをダウンロードする」を参照してください。

Systems Manager を使用してダウンロードする

Systems Manager Run Command を使用するには、オンプレミスサーバーを Amazon EC2 Systems Manager に登録する必要があります。詳細については、AWS Systems Manager ユーザーガイドのハイブリッド環境での Systems Manager のセットアップを参照してください。

既にサーバーを登録している場合、SSM Agent を最新バージョンに更新します。

Linux を実行しているサーバーでの SSM Agent の更新の詳細については、AWS Systems Manager ユーザーガイドのハイブリッド環境で SSM Agent をインストールする (Linux) を参照してください。

Windows Server を実行しているサーバーで SSM Agent を更新する方法の詳細については、AWS Systems Manager ユーザーガイドのハイブリッド環境で SSM Agent をインストールする (Windows) を参照してください。

SSM Agent を使用してオンプレミスサーバーで CloudWatch エージェントパッケージをダウンロードするには

1. Systems Manager コンソール (https://console.aws.amazon.com/systems-manager/) を開きます。
2. ナビゲーションペインで、[Run Command] を選択します。

   -または-

   AWS Systems Manager のホームページが表示された場合は、スクロールダウンして [Explore Run Command] (Run Command の詳細) を選択します。
3. [Run command (コマンドの実行)] を選択します。
4. [Command document] リストで、[AWS-ConfigureAWSPackage] の横のボタンを選択します。
5. [ターゲット] 領域で、CloudWatch エージェントをインストールするサーバーを選択します。特定のサーバーが表示されない場合、Run Command 用に設定されていない可能性があります。詳細については、AWS Systems Manager ユーザーガイドからハイブリッド環境での AWS Systems Manager のセットアップを参照してください。
6. [アクション] リストで、[インストール] を選択します。
7. [名前] ボックスに、「*AmazonCloudWatchAgent*」と入力します。
8. エージェントの最新バージョンをインストールには、[Version] を空白のままにします。
9. [Run (実行)] を選択します。

   エージェントパッケージがダウンロードされたら、次のステップではエージェントパッケージを設定して開始します。

(オンプレミスサーバーへのインストール) IAM 認証情報と AWS リージョンを指定する

CloudWatch エージェントで、オンプレミスサーバーのデータを送信できるようにするには、先ほど作成した IAM ユーザーのアクセスキーおよびシークレットキーを指定する必要があります。このユーザーの作成方法については、「CloudWatch エージェントで使用する IAM ロールとユーザーを作成する」を参照してください。

また、メトリクスを送信する AWS リージョンを指定する必要があります。これには、`region` フィールドを使用します。

このファイルの例を以下に示します。

```
[AmazonCloudWatchAgent]
```

```
aws_access_key_id=my_access_key
aws_secret_access_key=my_secret_key
region = us-west-1
```

my_access_key および *my_secret_key* で、Systems Manager Parameter Store に書き込むアクセス許可がない IAM ユーザーのキーを使用します。CloudWatch エージェントに必要な IAM ユーザーの詳細については、「オンプレミスサーバーで CloudWatch エージェントを使用するための IAM ユーザーを作成する」を参照してください。

このプロファイルの名前を AmazonCloudWatchAgent とする場合は、これ以上何もする必要はありません。必要に応じて、別の名前をつけ、shared_credential_profile ファイルの common-config.toml の値としてその名前を指定することができます。詳細は次のセクションを参照してください。

次の例では、aws configure コマンドを使用して CloudWatch エージェントの名前付きプロファイルを作成します。この例では、AmazonCloudWatchAgent のデフォルトプロファイル名を使用しているものとします。

CloudWatch エージェントの AmazonCloudWatchAgent プロファイルを作成するには

1. まだの場合は、AWS Command Line Interface をサーバーにインストールしてください。詳細については、「AWS CLI のインストール」を参照してください。
2. Linux サーバーでは、以下のコマンドを入力し、プロンプトに従います。

   ```
   sudo aws configure --profile AmazonCloudWatchAgent
   ```

   Windows Server では、管理者として PowerShell を開き、以下のコマンドを入力して、プロンプトに従います。

   ```
   aws configure --profile AmazonCloudWatchAgent
   ```

(オプション) CloudWatch エージェントの一般的な設定と名前付きプロファイルの変更

CloudWatch エージェントには、common-config.toml と呼ばれる設定ファイルが含まれます。必要に応じてこのファイルを使用して、プロキシおよびリージョン情報を指定できます。

Linux を実行しているサーバーでは、このファイルは /opt/aws/amazon-cloudwatch-agent/etc ディレクトリにあります。Windows Server を実行しているサーバーでは、このファイルは C:\ProgramData\Amazon\AmazonCloudWatchAgent ディレクトリにあります。

デフォルトの common-config.toml は次のようになります。

```
# This common-config is used to configure items used for both ssm and cloudwatch access


## Configuration for shared credential.
## Default credential strategy will be used if it is absent here:
##            Instance role is used for EC2 case by default.
##            AmazonCloudWatchAgent profile is used for onPremise case by default.
# [credentials]
#    shared_credential_profile = "{profile_name}"
#    shared_credential_file= "{file_name}"

## Configuration for proxy.
## System-wide environment-variable will be read if it is absent here.
## i.e. HTTP_PROXY/http_proxy; HTTPS_PROXY/https_proxy; NO_PROXY/no_proxy
## Note: system-wide environment-variable is not accessible when using ssm run-command.
## Absent in both here and environment-variable means no proxy will be used.
# [proxy]
#    http_proxy = "{http_url}"
#    https_proxy = "{https_url}"
#    no_proxy = "{domain}"
```

初期状態では、すべての行がコメントアウトされています。認証情報プロファイルやプロキシ設定を設定するには、その行から # を削除し、値を指定します。このファイルを手動で編集するか、Systems Manager の RunShellScript Run Command を使用できます。

- shared_credential_profile – オンプレミスサーバーの場合、このコード行では、データを CloudWatch に送信するために使用する IAM ユーザー認証情報を指定します。このコード行をコメントアウトした場合、AmazonCloudWatchAgent が使用されます。このプロファイルの作成の詳細については、「(オンプレミスサーバーへのインストール) IAM 認証情報と AWS リージョンを指定する」を参照してください。

  EC2 インスタンスでは、このコード行を使用して、CloudWatch エージェントでこのインスタンスのデータを別の AWS リージョンの CloudWatch に送信します。そのためには、送信先のリージョンの名前を指定する region フィールドを含む名前付きプロファイルを指定します。

  shared_credential_profile を指定する場合は、[credentials] 行の先頭から # を削除する必要もあります。

- shared_credential_file – エージェントでデフォルトのパス以外のパスにあるファイルの認証情報を検索するには、ここで完全なパスとファイル名を指定します。デフォルトのパスは、/root/.aws (Linux) および C:\\Users\\Administrator\\.aws (Windows Server) です。

  以下の最初の例は、Linux サーバーで有効な shared_credential_file 行の構文です。2 番目の例は、Windows Server で有効です。Windows Server では、\ 文字をエスケープする必要があります。

  ```
  shared_credential_file= "/usr/username/credentials"
  ```

  ```
  shared_credential_file= "C:\\Documents and Settings\\username\\.aws\\credentials"
  ```

  shared_credential_file を指定する場合は、[credentials] 行の先頭から # を削除する必要もあります。
- プロキシ設定 – サーバーで HTTP プロキシまたは HTTPS プロキシを使用して AWS サービスに接続する場合は、これらのプロキシを http_proxy フィールドと https_proxy フィールドに指定します。プロキシから除外すべき URL がある場合、カンマで区切って no_proxy フィールドで指定します。

CloudWatch エージェントの起動

CloudWatch エージェントは、Systems Manager Run Command またはコマンドラインを使用して開始できます。

SSM Agent を使用して、オンプレミスサーバーで CloudWatch エージェントを開始するには

1. Systems Manager コンソール (https://console.aws.amazon.com/systems-manager/) を開きます。
2. ナビゲーションペインで、[Run Command] を選択します。

   -または-

   AWS Systems Manager のホームページが表示された場合は、スクロールダウンして [Explore Run Command] (Run Command の詳細) を選択します。
3. [Run command (コマンドの実行)] を選択します。
4. [Command document] リストで、[AmazonCloudWatch-ManageAgent] の横のボタンを選択します。

5. [Targets] 領域で、エージェントをインストールしたインスタンスを選択します。
6. [Action] リストで、[configure] を選択します。
7. [Mode] リストで、[onPremise] を選択します。
8. [Optional Configuration Location (オプションの設定場所)] ボックスで、ウィザードで作成して Parameter Store に保存したエージェント設定ファイルの名前を入力します。
9. [Run (実行)] を選択します。

   エージェントが、設定ファイルで指定した設定で開始されます。

コマンドラインを使用してオンプレミスサーバーで CloudWatch エージェントを開始するには

- このコマンドで、`-a fetch-config` によりエージェントは最新バージョンの CloudWatch エージェント設定ファイルをロードし、`-s` がエージェントを開始します。

  Linux: Systems Manager Parameter Store に設定ファイルを保存した場合は、次のように入力します。

  ```
  sudo /opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-ctl -a fetch-
  config -m onPremise -s -c ssm:configuration-parameter-store-name
  ```

  Linux: 設定ファイルをローカルコンピュータに保存した場合は、次のコマンドを入力します。*configuration-file-path* は、エージェント設定ファイルへのパスに置き換えます。このファイルの名前は、ウィザードで作成した場合は `config.json` になり、手動で作成した場合は `amazon-cloudwatch-agent.json` になる場合があります。

  ```
  sudo /opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-ctl -a fetch-
  config -m onPremise -s -c file:configuration-file-path
  ```

  Windows Server: エージェント設定ファイルを Systems Manager Parameter Store に保存した場合は、PowerShell コンソールから次のように入力します。

  ```
  & "C:\Program Files\Amazon\AmazonCloudWatchAgent\amazon-cloudwatch-agent-ctl.ps1" -
  a fetch-config -m onPremise -s -c ssm:configuration-parameter-store-name
  ```

  Windows Server: エージェント設定ファイルをローカルコンピュータに保存した場合は、PowerShell コンソールから次のように入力します。*configuration-file-path* は、エージェント設定ファイルへのパスに置き換えます。このファイルの名前は、ウィザード

で作成した場合は config.json になり、手動で作成した場合は amazon-cloudwatch-agent.json になる場合があります。

```
& "C:\Program Files\Amazon\AmazonCloudWatchAgent\amazon-cloudwatch-agent-ctl.ps1" -a fetch-config -m onPremise -s -c file:configuration-file-path
```

AWS CloudFormation を使用して新しいインスタンスに CloudWatch エージェントをインストールする

Amazon は、新しい Amazon EC2 インスタンスへの CloudWatch エージェントのインストールおよび更新に役立つ複数の AWS CloudFormation テンプレートを GitHub にアップロードしています。AWS CloudFormation の使用の詳細については、「AWS CloudFormation とは」を参照してください。

テンプレートの場所は、AWS CloudFormation を使用して Amazon CloudWatch エージェントを EC2 インスタンスにデプロイするです。この場所には、inline ディレクトリと ssm ディレクトリの両方が含まれます。各ディレクトリに Linux インスタンスと Windows インスタンスの両方のテンプレートが含まれています。

- inline ディレクトリのテンプレートの場合は、CloudWatch エージェント設定が AWS CloudFormation テンプレートに埋め込まれています。デフォルトでは、Linux テンプレートが mem_used_percent メトリクスと swap_used_percent メトリクスを収集し、Windows テンプレートが Memory % Committed Bytes In Use メトリクスと Paging File % Usage メトリクスを収集します。

  さまざまなメトリクスが収集されるようにこれらのテンプレートを変更するには、テンプレートの次のセクションを変更します。Linux サーバーのテンプレートの例を次に示します。これらの変更を行うには、エージェント設定ファイルの形式と構文に従ってください。詳細については、「CloudWatch エージェント設定ファイルを手動で作成または編集する」を参照してください。

```
{
   "metrics":{
      "append_dimensions":{
         "AutoScalingGroupName":"${!aws:AutoScalingGroupName}",
         "ImageId":"${!aws:ImageId}",
         "InstanceId":"${!aws:InstanceId}",
         "InstanceType":"${!aws:InstanceType}"
      },
```

```
      "metrics_collected":{
         "mem":{
            "measurement":[
               "mem_used_percent"
            ]
         },
         "swap":{
            "measurement":[
               "swap_used_percent"
            ]
         }
      }
   }
}
```

> **Note**
>
> inline テンプレートの場合、すべてのプレースホルダ変数の先頭にエスケープ文字として感嘆符 (!) が必要です。これは、サンプルテンプレートで確認できます。他のプレースホルダ変数を追加する場合は、変数名の前に感嘆符を追加します。

- `ssm` ディレクトリのテンプレートでは、エージェント設定ファイルを Parameter Store からロードします。これらのテンプレートを使用するには、最初に設定ファイルを作成して Parameter Store にアップロードする必要があります。次に、このファイルの Parameter Store 名をテンプレート内で指定します。設定ファイルは手動で作成するか、ウィザードで作成できます。詳細については、「CloudWatch エージェント設定ファイルを作成する」を参照してください。

CloudWatch エージェントのインストールとエージェント設定の更新には、両方のタイプのテンプレートを使用できます。

チュートリアル: AWS CloudFormation inline テンプレートを使用して CloudWatch エージェントをインストールおよび設定する

このチュートリアルでは、AWS CloudFormation を使用して新しい Amazon EC2 インスタンスに CloudWatch エージェントをインストールする手順について説明します。このチュートリアルでは、inline テンプレートを使用して Amazon Linux 2 を実行する新しいインスタンスにインストールします。この場合、JSON 設定ファイルや Parameter Store を使用する必要はありません。inline テンプレートの場合、テンプレートにエージェント設定が含まれています。このチュートリアルでは、テンプレートに含まれているデフォルトのエージェント設定を使用します。

エージェントのインストール手順を示した後で、エージェントの更新方法を示します。

AWS CloudFormation を使用して新しいインスタンスに CloudWatch エージェントをインストールするには

1. GitHub からテンプレートをダウンロードします。このチュートリアルでは、次のように Amazon Linux 2 の inline テンプレートをダウンロードします。

   ```
   curl -O https://raw.githubusercontent.com/awslabs/aws-cloudformation-templates/
   master/aws/solutions/AmazonCloudWatchAgent/inline/amazon_linux.template
   ```

2. https://console.aws.amazon.com/cloudformation で AWS CloudFormation コンソール を開きます。
3. [Create stack] を選択します。
4. [テンプレートの選択] で、[テンプレートを Amazon S3 にアップロード] を選択し、ダウンロード済みのテンプレートを選択して [次へ] を選択します。
5. [詳細の指定] ページで、以下のパラメータを指定し、[次へ] を選択します。

   - Stack name: AWS CloudFormation スタックのスタック名を選択します。
   - IAMRole: CloudWatch メトリクス、ログ、トレースを書き込むアクセス許可を持つ IAM ロールを選択します。詳細については、「Amazon EC2 インスタンスの CloudWatch エージェントで使用する IAM ロールを作成する」を参照してください。
   - InstanceAMI: スタックを起動するリージョンで有効な AMI を選択します。
   - InstanceType: 有効なインスタンスタイプを選択します。
   - KeyName: 新しいインスタンスへの SSH アクセスを有効にするには、既存の Amazon EC2 キーペアを選択します。Amazon EC2 キーペアをまだ持っていない場合は、 で作成できます AWS Management Console 詳細については、Linux インスタンス用 Amazon EC2 ユーザーガイドの「Amazon EC2 のキーペア」を参照してください。
   - SSHLocation: SSH を使用してインスタンスに接続するために使用できる IP アドレス範囲を指定します。デフォルトでは、任意の IP アドレスからのアクセスを許可します。
6. [オプション] ページで、スタックリソースをタグ付けできます。[Next] を選択します。
7. [レビュー] ページで、情報を確認して、スタックで IAM リソースが作成されることを認識し、[作成] を選択します。

   コンソールを更新すると、新しいスタックが CREATE_IN_PROGRESS スタータスで表示されます。

8. インスタンスが作成されると、Amazon EC2 コンソールに表示されます。必要に応じてホストに接続し、進行状況をチェックできます。

   エージェントがインストールされたことを確認するには、次のコマンドを使用します。

   ```
   rpm -qa amazon-cloudwatch-agent
   ```

   エージェントが実行中であることを確認するには、次のコマンドを使用します。

   ```
   ps aux | grep amazon-cloudwatch-agent
   ```

次の手順では、inline テンプレートを使用して AWS CloudFormation で CloudWatch エージェントを更新する方法を示します。デフォルトの inline テンプレートでは mem_used_percent メトリクスを収集します。このチュートリアルでは、このメトリクスの収集を停止するようにエージェント設定を変更します。

AWS CloudFormation を使用して CloudWatch エージェントを更新するには

1. 前の手順でダウンロードしたテンプレートで、以下の行を削除し、テンプレートを保存します。

   ```
   "mem": {

        "measurement": [
            "mem_used_percent"
          ]
    },
   ```

2. https://console.aws.amazon.com/cloudformation で AWS CloudFormation コンソール を開きます。
3. AWS CloudFormation ダッシュボードで、作成したスタックを選択後、[スタックの更新] を選択します。
4. [テンプレートの選択] で、[テンプレートを Amazon S3 にアップロード] を選択し、変更したテンプレートを選択して [次へ] を選択します。
5. [オプション] ページで、[次へ]、[次へ] の順に選択します。
6. [確認] ページで情報を確認し、[更新] を選択します。

   しばらくすると、UPDATE_COMPLETE と表示されます。

チュートリアル: AWS CloudFormation と Parameter Store を使用して CloudWatch エージェントをインストールする

このチュートリアルでは、AWS CloudFormation を使用して新しい Amazon EC2 インスタンスに CloudWatch エージェントをインストールする手順について説明します。このチュートリアルでは、Parameter Store で作成して保存したエージェント設定ファイルを使用して、Amazon Linux 2 を実行する新しいインスタンスにインストールします。

エージェントのインストール手順を示した後で、エージェントの更新方法を示します。

AWS CloudFormation を使用して、Parameter Store の設定により CloudWatch エージェントを新しいインスタンスにインストールするには

1. まだ行っていない場合は、CloudWatch エージェントパッケージをコンピュータにダウンロードし、エージェント設定ファイルを作成できるようにします。Parameter Store を使用したエージェントの詳細とダウンロードについては、「CloudWatch エージェントをダウンロードして設定する」を参照してください。コマンドラインを使用したパッケージの詳細とダウンロードについては、「コマンドラインを使用して CloudWatch エージェントをダウンロードおよび設定する」を参照してください。
2. Parameter Store でエージェント設定ファイルを作成して保存します。詳細については、「CloudWatch エージェント設定ファイルを作成する」を参照してください。
3. 次のように、GitHub からテンプレートをダウンロードします。

   ```
   curl -O https://raw.githubusercontent.com/awslabs/aws-cloudformation-templates/
   master/aws/solutions/AmazonCloudWatchAgent/ssm/amazon_linux.template
   ```

4. https://console.aws.amazon.com/cloudformation で AWS CloudFormation コンソール を開きます。
5. [Create stack] を選択します。
6. [テンプレートの選択] で、[テンプレートを Amazon S3 にアップロード] を選択し、ダウンロードしたテンプレートを選択して [次へ] を選択します。
7. [詳細の指定] ページで、必要に応じて以下のパラメータを指定し、[次へ] を選択します。

   - Stack name: AWS CloudFormation スタックのスタック名を選択します。
   - IAMRole: CloudWatch メトリクス、ログ、トレースを書き込むアクセス許可を持つ IAM ロールを選択します。詳細については、「Amazon EC2 インスタンスの CloudWatch エージェントで使用する IAM ロールを作成する」を参照してください。

- InstanceAMI: スタックを起動するリージョンで有効な AMI を選択します。
- InstanceType: 有効なインスタンスタイプを選択します。
- KeyName: 新しいインスタンスへの SSH アクセスを有効にするには、既存の Amazon EC2 キーペアを選択します。Amazon EC2 キーペアをまだ持っていない場合は、 で作成できます AWS Management Console 詳細については、Linux インスタンス用 Amazon EC2 ユーザーガイドの「Amazon EC2 のキーペア」を参照してください。
- SSHLocation: SSH を使用してインスタンスに接続するために使用できる IP アドレス範囲を指定します。デフォルトでは、任意の IP アドレスからのアクセスを許可します。
- SSMKey: Parameter Store で作成して保存したエージェント設定ファイルを指定します。

8. [オプション] ページで、スタックリソースをタグ付けできます。[Next] を選択します。
9. [レビュー] ページで、情報を確認して、スタックで IAM リソースが作成されることを認識し、[作成] を選択します。

   コンソールを更新すると、新しいスタックが CREATE_IN_PROGRESS スタータスで表示されます。
10. インスタンスが作成されると、Amazon EC2 コンソールに表示されます。必要に応じてホストに接続し、進行状況をチェックできます。

    エージェントがインストールされたことを確認するには、次のコマンドを使用します。

    ```
    rpm -qa amazon-cloudwatch-agent
    ```

    エージェントが実行中であることを確認するには、次のコマンドを使用します。

    ```
    ps aux | grep amazon-cloudwatch-agent
    ```

次の手順では、Parameter Store に保存したエージェント設定を使用して、AWS CloudFormation により CloudWatch エージェントを更新する方法を示します。

AWS CloudFormation を使用して、Parameter Store の設定により CloudWatch エージェントを更新するには

1. 必要に応じて、Parameter Store に保存したエージェント設定ファイルを新しい設定に変更します。

2. AWS CloudFormation トピックでダウンロードした the section called “チュートリアル: AWS CloudFormation と Parameter Store を使用して CloudWatch エージェントをインストールする” テンプレートで、バージョン番号を変更します。たとえば、`VERSION=1.0` を `VERSION=2.0` に変更できます。
3. https://console.aws.amazon.com/cloudformation で AWS CloudFormation コンソール を開きます。
4. AWS CloudFormation ダッシュボードで、作成したスタックを選択後、[スタックの更新] を選択します。
5. [テンプレートの選択] で、[テンプレートを Amazon S3 にアップロード] を選択し、変更したテンプレートを選択して [次へ] を選択します。
6. [オプション] ページで、[次へ]、[次へ] の順に選択します。
7. [確認] ページで情報を確認し、[更新] を選択します。

   しばらくすると、`UPDATE_COMPLETE` と表示されます。

AWS CloudFormation での CloudWatch エージェントのインストールのトラブルシューティング

このセクションは、 AWS CloudFormationによる CloudWatch エージェントのインストールおよび更新に関する問題のトラブルシューティングに役立ちます。

更新が失敗した場合の検出

AWS CloudFormation を使用して CloudWatch エージェントの設定を更新し、無効な設定を使用すると、エージェントは CloudWatch へのあらゆるメトリクスの送信を停止します。エージェント設定が正常に更新されたかどうかは、`cfn-init-cmd.log` ファイルで簡単に確認できます。Linux サーバーの場合、このファイルは `/var/log/cfn-init-cmd.log` にあります。Windows インスタンスの場合、このファイルは `C:\cfn\log\cfn-init-cmd.log` にあります。

メトリクスが見つからない

エージェントのインストールまたは更新後に必要なメトリクスが表示されない場合は、そのメトリクスを収集するようにエージェントが設定されていることを確認します。そのためには、`amazon-cloudwatch-agent.json` ファイルにメトリクスが表示されていることと、正しいメトリクスの名前空間を参照していることを確認します。詳細については、「CloudWatch エージェントファイルとロケーション」を参照してください。

CloudWatch エージェント認証情報の優先設定

このセクションでは、CloudWatch エージェントが他の AWS サービスや API と通信する際に認証情報を取得するために使用する認証情報プロバイダーチェーンの概要を説明します。順序は次のとおりです。以下のリストの 2 番目から 5 番目までの設定は、AWS SDK で定義されている優先順位と同じです。詳細については、SDK ドキュメントの「認証情報の指定」を参照してください。

1. CloudWatch エージェントの `common-config.toml` ファイルに定義されている共有設定ファイルおよび認証情報ファイル。詳細については、「(オプション) プロキシ情報またはリージョン情報の一般的な設定を変更する」を参照してください。
2. AWS SDK 環境変数

> **Important**
>
> Linux では、`amazon-cloudwatch-agent-ctl` スクリプトを使用して CloudWatch エージェントを実行する場合、スクリプトはエージェントを `systemd` サービスとして起動します。この場合、`HOME`、`AWS_ACCESS_KEY_ID`、`AWS_SECRET_ACCESS_KEY` などの環境変数にはエージェントからアクセスできません。

3. `$HOME/%USERPROFILE%` にある共有設定ファイルおよび認証情報ファイル。

> **Note**
>
> CloudWatch エージェントは、Linux と MacOS の場合は `$HOME` の `.aws/credentials` を検索し、Windows の場合は `%USERPROFILE%` を検索します。AWS SDK とは異なり、CloudWatch エージェントには、環境変数にアクセスできない場合にホームディレクトリを決定するフォールバックメソッドはありません。この動作の違いは、AWS SDK の以前の実装との下位互換性を維持するためです。
> さらに、`common-config.toml` にある共有認証情報とは異なり、AWS SDK から派生した共有認証情報が期限切れになってローテーションされても、更新された認証情報は CloudWatch エージェントによって自動的に取得されないため、エージェントの再起動が必要になります。

4. Amazon Elastic Container Service タスク定義または RunTask API オペレーションを使用するアプリケーションが存在する場合、タスクの AWS Identity and Access Management ロール。
5. Amazon EC2 インスタンスにアタッチされたインスタンスプロファイル。

ベストプラクティスとして、CloudWatch エージェントを使用するときは、次の順序で認証情報を指定することをお勧めします。

1. アプリケーションが Amazon Elastic Container Service タスク定義または RunTask API オペレーションを使用している場合は、タスクの IAM ロールを使用します。
2. アプリケーションが Amazon EC2 インスタンスで実行される場合は、IAM ロールを使用します。
3. CloudWatch エージェントの `common-config.toml` ファイルを使用して認証情報ファイルを指定します。この認証情報ファイルと同じものが他の AWS SDK や AWS CLI でも使用されます。共有の認証情報ファイルを既に使用している場合は、そのファイルを当目的にも使用できます。CloudWatch エージェントの `common-config.toml` ファイルを使用して提供する場合、認証情報の有効期限が切れて置き換えられるときに、エージェントを再起動しなくてもローテーションされた認証情報をエージェントが使用できるようになります。
4. 環境変数を使用します。Amazon EC2 インスタンス以外のコンピュータで開発作業を行う場合は、環境変数を設定すると便利です。

> **Note**
>
> 「別のアカウントへのメトリクス、ログ、トレースの送信」で説明されているように、テレメトリを別のアカウントに送信する場合、CloudWatch エージェントは、このセクションで説明する認証情報プロバイダーチェーンを使用して、認証情報の初期セットを取得します。その後、CloudWatch エージェント設定ファイルの `role_arn` で指定された IAM ロールを引き受けるときに、これらの認証情報を使用します。

CloudWatch エージェントパッケージの署名の検証

GPG 署名ファイルは、Linux サーバー上の CloudWatch エージェントパッケージ用として含まれています。公開鍵を使用して、エージェントのダウンロードファイルがオリジナルであり、変更されていないことを確認できます。

Windows Server の場合は、MSI を使用して署名を検証できます。

macOS コンピュータの場合、署名はエージェントのダウンロードパッケージに含まれます。

正しい署名ファイルを検索するには、次の表を参照してください。各アーキテクチャおよびオペレーティングシステムごとに、全般的なリンクおよび各リージョンのリンクがあります。例えば、Amazon Linux 2023、Amazon Linux 2、x86-64 アーキテクチャの場合、有効なリンクは次の 3 つです。

- `https://amazoncloudwatch-agent.s3.amazonaws.com/amazon_linux/amd64/latest/amazon-cloudwatch-agent.rpm.sig`
- `https://amazoncloudwatch-agent-us-east-1.s3.us-east-1.amazonaws.com/amazon_linux/amd64/latest/amazon-cloudwatch-agent.rpm`
- `https://amazoncloudwatch-agent-eu-central-1.s3.eu-central-1.amazonaws.com/amazon_linux/amd64/latest/amazon-cloudwatch-agent.rpm`

> **Note**
>
> CloudWatch エージェントをダウンロードするには、接続に TLS 1.2 以降を使用する必要があります。

アーキテクチャ	プラットフォーム	ダウンロードリンク	署名ファイルリンク
x86-64	Amazon Linux 2023 および Amazon Linux 2	https://amazoncloudwatch-agent.s3.amazonaws.com/amazon_linux/amd64/latest/amazon-cloudwatch-agent.rpm https://amazoncloudwatch-agent-*region*.s3.*region*.amazonaws.com/amazon_linux/amd64/latest/amazon-cloudwatch-agent.rpm	https://amazoncloudwatch-agent.s3.amazonaws.com/amazon_linux/amd64/latest/amazon-cloudwatch-agent.rpm.sig https://amazoncloudwatch-agent-*region*.s3.*region*.amazonaws.com/amazon_linux/amd64/latest/amazon-cloudwatch-agent.rpm.sig
x86-64	CentOS	https://amazoncloudwatch-agent.s3.amazonaws.com/centos/amd64/latest/amazon-cloudwatch-agent.rpm https://amazoncloudwatch-agent-*region*.s3.*region*.amazon	https://amazoncloudwatch-agent.s3.amazonaws.com/centos/amd64/latest/amazon-cloudwatch-agent.rpm.sig https://amazoncloudwatch-agent-*region*.s3.*region*.amazonaw

アーキテクチャ	プラットフォーム	ダウンロードリンク	署名ファイルリンク
		s.com/centos/amd64/latest/amazon-cloudwatch-agent.rpm	s.com/centos/amd64/latest/amazon-cloudwatch-agent.rpm.sig
x86-64	Redhat	https://amazoncloudwatch-agent.s3.amazonaws.com/redhat/amd64/latest/amazon-cloudwatch-agent.rpm https://amazoncloudwatch-agent-*region*.s3.*region*.amazonaws.com/redhat/amd64/latest/amazon-cloudwatch-agent.rpm	https://amazoncloudwatch-agent.s3.amazonaws.com/redhat/amd64/latest/amazon-cloudwatch-agent.rpm.sig https://amazoncloudwatch-agent-*region*.s3.*region*.amazonaws.com/redhat/amd64/latest/amazon-cloudwatch-agent.rpm.sig
x86-64	SUSE	https://amazoncloudwatch-agent.s3.amazonaws.com/suse/amd64/latest/amazon-cloudwatch-agent.rpm https://amazoncloudwatch-agent-*region*.s3.*region*.amazonaws.com/suse/amd64/latest/amazon-cloudwatch-agent.rpm	https://amazoncloudwatch-agent.s3.amazonaws.com/suse/amd64/latest/amazon-cloudwatch-agent.rpm.sig https://amazoncloudwatch-agent-*region*.s3.*region*.amazonaws.com/suse/amd64/latest/amazon-cloudwatch-agent.rpm.sig
x86-64	Debian	https://amazoncloudwatch-agent.s3.amazonaws.com/debian/amd64/latest/amazon-cloudwatch-agent.deb https://amazoncloudwatch-agent-*region*.s3.*region*.amazonaws.com/debian/amd64/latest/amazon-cloudwatch-agent.deb	https://amazoncloudwatch-agent.s3.amazonaws.com/debian/amd64/latest/amazon-cloudwatch-agent.deb.sig https://amazoncloudwatch-agent-*region*.s3.*region*.amazonaws.com/debian/amd64/latest/amazon-cloudwatch-agent.deb.sig

アーキテクチャ	プラットフォーム	ダウンロードリンク	署名ファイルリンク
x86-64	Ubuntu	https://amazoncloudwatch-agent.s3.amazonaws.com/ubuntu/amd64/latest/amazon-cloudwatch-agent.deb https://amazoncloudwatch-agent-*region*.s3.*region*.amazonaws.com/ubuntu/amd64/latest/amazon-cloudwatch-agent.deb	https://amazoncloudwatch-agent.s3.amazonaws.com/ubuntu/amd64/latest/amazon-cloudwatch-agent.deb.sig https://amazoncloudwatch-agent-*region*.s3.*region*.amazonaws.com/ubuntu/amd64/latest/amazon-cloudwatch-agent.deb.sig
x86-64	Oracle	https://amazoncloudwatch-agent.s3.amazonaws.com/oracle_linux/amd64/latest/amazon-cloudwatch-agent.rpm https://amazoncloudwatch-agent-*region*.s3.*region*.amazonaws.com/oracle_linux/amd64/latest/amazon-cloudwatch-agent.rpm	https://amazoncloudwatch-agent.s3.amazonaws.com/oracle_linux/amd64/latest/amazon-cloudwatch-agent.rpm.sig https://amazoncloudwatch-agent-*region*.s3.*region*.amazonaws.com/oracle_linux/amd64/latest/amazon-cloudwatch-agent.rpm.sig
x86-64	macOS	https://amazoncloudwatch-agent.s3.amazonaws.com/darwin/amd64/latest/amazon-cloudwatch-agent.pkg https://amazoncloudwatch-agent-*region*.s3.*region*.amazonaws.com/darwin/amd64/latest/amazon-cloudwatch-agent.pkg	https://amazoncloudwatch-agent.s3.amazonaws.com/darwin/amd64/latest/amazon-cloudwatch-agent.pkg.sig https://amazoncloudwatch-agent-*region*.s3.*region*.amazonaws.com/darwin/amd64/latest/amazon-cloudwatch-agent.pkg.sig

アーキテクチャ	プラットフォーム	ダウンロードリンク	署名ファイルリンク
x86-64	Windows	https://amazoncloudwatch-agent.s3.amazonaws.com/windows/amd64/latest/amazon-cloudwatch-agent.msi https://amazoncloudwatch-agent-*region*.s3.*region*.amazonaws.com/windows/amd64/latest/amazon-cloudwatch-agent.msi	https://amazoncloudwatch-agent.s3.amazonaws.com/windows/amd64/latest/amazon-cloudwatch-agent.msi.sig https://amazoncloudwatch-agent-*region*.s3.*region*.amazonaws.com/windows/amd64/latest/amazon-cloudwatch-agent.msi.sig
ARM64	Amazon Linux 2023 および Amazon Linux 2	https://amazoncloudwatch-agent.s3.amazonaws.com/amazon_linux/arm64/latest/amazon-cloudwatch-agent.rpm https://amazoncloudwatch-agent-*region*.s3.*region*.amazonaws.com/amazon_linux/arm64/latest/amazon-cloudwatch-agent.rpm	https://amazoncloudwatch-agent.s3.amazonaws.com/amazon_linux/arm64/latest/amazon-cloudwatch-agent.rpm.sig https://amazoncloudwatch-agent-*region*.s3.*region*.amazonaws.com/amazon_linux/arm64/latest/amazon-cloudwatch-agent.rpm.sig
ARM64	Red Hat	https://amazoncloudwatch-agent.s3.amazonaws.com/redhat/arm64/latest/amazon-cloudwatch-agent.rpm https://amazoncloudwatch-agent-*region*.s3.*region*.amazonaws.com/redhat/arm64/latest/amazon-cloudwatch-agent.rpm	https://amazoncloudwatch-agent.s3.amazonaws.com/redhat/arm64/latest/amazon-cloudwatch-agent.rpm.sig https://amazoncloudwatch-agent-*region*.s3.*region*.amazonaws.com/redhat/arm64/latest/amazon-cloudwatch-agent.rpm.sig

アーキテクチャ	プラットフォーム	ダウンロードリンク	署名ファイルリンク
ARM64	Ubuntu	https://amazoncloudwatch-agent.s3.amazonaws.com/ubuntu/arm64/latest/amazon-cloudwatch-agent.deb https://amazoncloudwatch-agent-*region*.s3.*region*.amazonaws.com/ubuntu/arm64/latest/amazon-cloudwatch-agent.deb	https://amazoncloudwatch-agent.s3.amazonaws.com/ubuntu/arm64/latest/amazon-cloudwatch-agent.deb.sig https://amazoncloudwatch-agent-*region*.s3.*region*.amazonaws.com/ubuntu/arm64/latest/amazon-cloudwatch-agent.deb.sig
ARM64	SUSE	https://amazoncloudwatch-agent.s3.amazonaws.com/suse/arm64/latest/amazon-cloudwatch-agent.rpm https://amazoncloudwatch-agent-*region*.s3.*region*.amazonaws.com/suse/arm64/latest/amazon-cloudwatch-agent.rpm	https://amazoncloudwatch-agent.s3.amazonaws.com/suse/arm64/latest/amazon-cloudwatch-agent.rpm.sig https://amazoncloudwatch-agent-*region*.s3.*region*.amazonaws.com/suse/arm64/latest/amazon-cloudwatch-agent.rpm.sig

Linux サーバーで CloudWatch エージェントパッケージを確認するには

1. 公開鍵をダウンロードします。

```
shell$ wget https://amazoncloudwatch-agent.s3.amazonaws.com/assets/amazon-
cloudwatch-agent.gpg
```

2. 公開鍵をキーリングにインポートします。

```
shell$  gpg --import amazon-cloudwatch-agent.gpg
gpg: key 3B789C72: public key "Amazon CloudWatch Agent" imported
gpg: Total number processed: 1
gpg: imported: 1 (RSA: 1)
```

次の手順で必要になるため、キーの値を書きとめておきます。前述の例では、キーの値は 3B789C72 です。

3. 次のコマンドを使用してフィンガープリントを確認し、*###*を前述の手順の値と置き換えます。

```
shell$  gpg --fingerprint key-value
pub   2048R/3B789C72 2017-11-14
      Key fingerprint = 9376 16F3 450B 7D80 6CBD  9725 D581 6730 3B78 9C72
uid                  Amazon CloudWatch Agent
```

フィンガープリントは、次のものと同一になる必要があります。

```
9376 16F3 450B 7D80 6CBD 9725 D581 6730 3B78 9C72
```

フィンガープリント文字列が一致しない場合は、エージェントをインストールしないでください。Amazon Web Services にお問い合わせください。

フィンガープリントを確認したら、それを使用して、CloudWatch エージェントパッケージの署名を確認できます。

4. wget を使用して、パッケージ署名ファイルをダウンロードします。正しい署名ファイルを確認するには、前の表を参照してください。

```
wget Signature File Link
```

5. 署名を確認するには、gpg --verify を実行します。

```
shell$ gpg --verify signature-filename agent-download-filename
gpg: Signature made Wed 29 Nov 2017 03:00:59 PM PST using RSA key ID 3B789C72
gpg: Good signature from "Amazon CloudWatch Agent"
gpg: WARNING: This key is not certified with a trusted signature!
gpg:          There is no indication that the signature belongs to the owner.
Primary key fingerprint: 9376 16F3 450B 7D80 6CBD  9725 D581 6730 3B78 9C72
```

出力結果に「BAD signature」という句が含まれる場合、手順が正しいことをもう一度確認してください。このレスポンスが続く場合は、Amazon Web Services に連絡し、ダウンロードしたファイルは使用しないでください。

信頼性に関する警告に注意します。キーは、自分や信頼する人が署名した場合にのみ信頼できます。つまり、署名が無効であるわけではなく、パブリックキーを確認していないことになります。

Windows Server を実行するサーバーで CloudWatch エージェントパッケージを確認するには

1. Windows 用の GnuPG を からダウンロードしてインストールしますhttps://gnupg.org/download/ インストールするときは、[Shell Extension (GpgEx)] オプションを含めます。

   残りの手順は、Windows PowerShell で実行できます。

2. 公開鍵をダウンロードします。

```
PS> wget https://amazoncloudwatch-agent.s3.amazonaws.com/assets/amazon-cloudwatch-agent.gpg -OutFile amazon-cloudwatch-agent.gpg
```

3. 公開鍵をキーリングにインポートします。

```
PS>  gpg --import amazon-cloudwatch-agent.gpg
gpg: key 3B789C72: public key "Amazon CloudWatch Agent" imported
gpg: Total number processed: 1
gpg: imported: 1 (RSA: 1)
```

   次の手順で必要になるため、キーの値を書きとめておきます。前述の例では、キーの値は 3B789C72 です。

4. 次のコマンドを使用してフィンガープリントを確認し、*###*を前述の手順の値と置き換えます。

```
PS>  gpg --fingerprint key-value
pub   rsa2048 2017-11-14 [SC]
      9376 16F3 450B 7D80 6CBD  9725 D581 6730 3B78 9C72
uid           [ unknown] Amazon CloudWatch Agent
```

   フィンガープリントは、次のものと同一になる必要があります。

   9376 16F3 450B 7D80 6CBD 9725 D581 6730 3B78 9C72

   フィンガープリント文字列が一致しない場合は、エージェントをインストールしないでください。Amazon Web Services にお問い合わせください。

   フィンガープリントを確認したら、それを使用して、CloudWatch エージェントパッケージの署名を確認できます。

5. wget を使用して、パッケージ署名ファイルをダウンロードします。正しい署名ファイルを確認するには、「CloudWatch エージェントダウンロードリンク」を参照してください。

6. 署名を確認するには、gpg --verify を実行します。

```
PS> gpg --verify sig-filename agent-download-filename
gpg: Signature made 11/29/17 23:00:45 Coordinated Universal Time
gpg:                using RSA key D58167303B789C72
gpg: Good signature from "Amazon CloudWatch Agent" [unknown]
gpg: WARNING: This key is not certified with a trusted signature!
gpg:          There is no indication that the signature belongs to the owner.
Primary key fingerprint: 9376 16F3 450B 7D80 6CBD  9725 D581 6730 3B78 9C72
```

出力結果に「BAD signature」という句が含まれる場合、手順が正しいことをもう一度確認してください。このレスポンスが続く場合は、Amazon Web Services に連絡し、ダウンロードしたファイルは使用しないでください。

信頼性に関する警告に注意します。キーは、自分や信頼する人が署名した場合にのみ信頼できます。つまり、署名が無効であるわけではなく、パブリックキーを確認していないことになります。

macOS コンピュータで CloudWatch エージェントパッケージを確認するには

- macOS での署名の検証には 2 つの方法があります。
  - 次のコマンドを実行して、フィンガープリントを検証します。

    ```
    pkgutil --check-signature amazon-cloudwatch-agent.pkg
    ```

    次のような結果が表示されます。

    ```
    Package "amazon-cloudwatch-agent.pkg":
           Status: signed by a developer certificate issued by Apple for
     distribution
           Signed with a trusted timestamp on: 2020-10-02 18:13:24 +0000
           Certificate Chain:
           1. Developer ID Installer: AMZN Mobile LLC (94KV3E626L)
           Expires: 2024-10-18 22:31:30 +0000
           SHA256 Fingerprint:
           81 B4 6F AF 1C CA E1 E8 3C 6F FB 9E 52 5E 84 02 6E 7F 17 21 8E FB
           0C 40 79 13 66 8D 9F 1F 10 1C

     ------------------------------------------------------------------------
           2. Developer ID Certification Authority
           Expires: 2027-02-01 22:12:15 +0000
    ```

```
        SHA256 Fingerprint:
        7A FC 9D 01 A6 2F 03 A2 DE 96 37 93 6D 4A FE 68 09 0D 2D E1 8D 03
        F2 9C 88 CF B0 B1 BA 63 58 7F

-----------------------------------------------------------------------
        3. Apple Root CA
        Expires: 2035-02-09 21:40:36 +0000
        SHA256 Fingerprint:
        B0 B1 73 0E CB C7 FF 45 05 14 2C 49 F1 29 5E 6E DA 6B CA ED 7E 2C
        68 C5 BE 91 B5 A1 10 01 F0 24
```

- または、.sig ファイルをダウンロードして使用する方法を使用するには、次の手順を実行します。
  - 次のコマンドを入力して、GPG アプリケーションを macOS ホストにインストールします。

    ```
    brew install GnuPG
    ```

- curl を使用して、パッケージ署名ファイルをダウンロードします。正しい署名ファイルを確認するには、「CloudWatch エージェントダウンロードリンク」を参照してください。
- 署名を確認するには、gpg --verify を実行します。

```
PS> gpg --verify sig-filename agent-download-filename
gpg: Signature made 11/29/17 23:00:45 Coordinated Universal Time
gpg:                using RSA key D58167303B789C72
gpg: Good signature from "Amazon CloudWatch Agent" [unknown]
gpg: WARNING: This key is not certified with a trusted signature!
gpg:          There is no indication that the signature belongs to the owner.
Primary key fingerprint: 9376 16F3 450B 7D80 6CBD  9725 D581 6730 3B78 9C72
```

出力結果に「BAD signature」という句が含まれる場合、手順が正しいことをもう一度確認してください。このレスポンスが続く場合は、Amazon Web Services に連絡し、ダウンロードしたファイルは使用しないでください。

信頼性に関する警告に注意します。キーは、自分や信頼する人が署名した場合にのみ信頼できます。つまり、署名が無効であるわけではなく、パブリックキーを確認していないことになります。

CloudWatch エージェント設定ファイルを作成する

CloudWatch エージェントをサーバーで実行する前に、CloudWatch エージェント設定ファイルを 1 つ以上作成する必要があります。

エージェント設定ファイルは、カスタムメトリクスを含め、エージェントが収集するメトリクス、ログ、トレースを指定する JSON ファイルです。このファイルを作成するには、ウィザードを使用するか、一から自分で作成します。また、ウィザードを使用して最初に設定ファイルを作成してから、手動で変更することもできます。ファイル手動で作成または変更するほうがプロセスが複雑になりますが、収集されるメトリクスをきめ細かくコントロールし、ウィザードで使用できないメトリクスを指定できます。

エージェント設定ファイルを変更するときは必ず、エージェントを再起動して変更を有効にする必要があります。エージェントを再起動するには、「CloudWatch エージェントを起動する」の手順に従います。

設定ファイルを作成したら、手動で JSON ファイルとして保存し、エージェントをサーバーにインストールする際に使用することができます。また、エージェントをサーバーにインストールする際に Systems Manager を使用する場合は、Systems Manager Parameter Store に保存することもできます。

CloudWatch エージェントは複数の設定ファイルの使用に対応しています。詳細については、「CloudWatch エージェント設定ファイル」を参照してください。

CloudWatch エージェントによって収集されたメトリクス、ログ、トレースには料金が発生します。料金の詳細については、「Amazon CloudWatch の料金」を参照してください。

内容

ウィザードを使用して CloudWatch エージェント設定ファイルを作成する

エージェント設定ファイルウィザード (`amazon-cloudwatch-agent-config-wizard`) では、ニーズに合わせて CloudWatch エージェントを設定するための一連の質問を行います。

必要な認証情報

ウィザードの開始前に AWS 認証方法および設定ファイルを所定の場所に置いた場合、ウィザードは使用する認証情報と AWS リージョンを自動検出できます。これらのファイルの詳細については、AWS Systems Manager ユーザーガイドの設定ファイルと認証情報ファイルを参照してください。

ウィザードでは、AWS 認証ファイルのデフォルトの認証情報を確認するとともに、次のような AmazonCloudWatchAgent セクションを探します。

```
[AmazonCloudWatchAgent]
aws_access_key_id = my_access_key
aws_secret_access_key = my_secret_key
```

また、このウィザードには、デフォルトの認証情報、AmazonCloudWatchAgent の認証情報、および Others オプションが表示されます。使用する認証情報を選択することができます。Others を選択した場合は、認証情報を入力できます。

my_access_key および *my_secret_key* で、Systems Manager Parameter Store に書き込むアクセス許可がある IAM ユーザーのキーを使用します。CloudWatch エージェントに必要な IAM ユーザーの詳細については、「オンプレミスサーバーで CloudWatch エージェントを使用するための IAM ユーザーを作成する」を参照してください。

AWS 設定ファイルで、エージェントからメトリクスを送信する先のリージョンを指定できます ([default] セクションのリージョンと異なる場合)。デフォルトでは、Amazon EC2 インスタンスが配置されているリージョンにメトリクスが発行されます。メトリクスを別のリージョンに発行する場合は、ここでリージョンを指定します。次の例では、メトリクスを us-west-1 リージョンに発行しています。

```
[AmazonCloudWatchAgent]
region = us-west-1
```

CloudWatch エージェント設定ウィザードを実行する

CloudWatch エージェント設定ファイルを作成するには

1. CloudWatch エージェント設定ウィザードを開始するには、次のように入力します。

   ```
   sudo /opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-config-wizard
   ```

Windows Server を実行しているサーバーで、次のコマンドを実行してウィザードを起動します。

```
cd "C:\Program Files\Amazon\AmazonCloudWatchAgent"
```

```
.\amazon-cloudwatch-agent-config-wizard.exe
```

2. 質問に答えて、サーバーの設定ファイルをカスタマイズします。
3. 設定ファイルをローカルに保存する場合、設定ファイル `config.json` は Linux サーバーでは `/opt/aws/amazon-cloudwatch-agent/bin/` に保存され、Windows Server では `C:\Program Files\Amazon\AmazonCloudWatchAgent` に保存されます。これで、エージェントをインストールする他のサーバーにこのファイルをコピーできます。

   Systems Manager を使用してエージェントをインストールおよび設定する場合、ファイルを Systems Manager Parameter Store に保存するかどうかを確認するよう求められたら、必ず「Yes」と回答してください。SSM Agent を使用して CloudWatch エージェントをインストールしていない場合でも、ファイルを Parameter Store に保存することもできます。Parameter Store にファイルを保存できるようにするには、十分なアクセス権限を持つ IAM ロールを使用する必要があります。詳細については、「CloudWatch エージェントで使用する IAM ロールとユーザーを作成する」を参照してください。

CloudWatch エージェントの事前定義されたメトリクスセット

ウィザードは、異なる詳細レベルを持つ、事前定義されたメトリクスセットで設定されます。これらのメトリクスのセットを次の表に示します。これらのメトリクスの詳細については、CloudWatch エージェントにより収集されるメトリクスを参照してください。

> **Note**
>
> Parameter Store では、標準とアドバンストの階層のパラメータをサポートしています。これらのパラメータ層は、これらの表で説明されているメトリクス詳細のベーシックレベル、標準レベル、アドバンストレベルとは関係ありません。

Linux を実行する Amazon EC2 インスタンス

詳細レベル	含められるメトリクス
Basic (ベーシック)	Mem: mem_used_percent Disk: disk_used_percent disk などの disk_used_percent メトリクスには Partition のディメンションがあります。つまり、生成されるカスタムメトリクスの数は、インスタンスに関連付けられたパーティションの数によって異なります。ディスクパーティションの数は、使用している AMI とサーバーにアタッチする Amazon EBS ボリュームの数によって異なります。
スタンダード	CPU: cpu_usage_idle 、cpu_usage_iowait 、cpu_usage_user 、cpu_usage_system Disk: disk_used_percent 、disk_inodes_free Diskio: diskio_io_time Mem: mem_used_percent Swap: swap_used_percent
アドバンスト	CPU: cpu_usage_idle 、cpu_usage_iowait 、cpu_usage_user 、cpu_usage_system Disk: disk_used_percent 、disk_inodes_free Diskio: diskio_io_time 、diskio_write_bytes 、diskio_read_bytes 、diskio_writes 、diskio_reads Mem: mem_used_percent Netstat: netstat_tcp_established 、netstat_tcp_time_wait Swap: swap_used_percent

Linux を実行しているオンプレミスサーバー

詳細レベル	含められるメトリクス
Basic (ベーシック)	Disk: disk_used_percent Diskio: diskio_write_bytes 、diskio_read_bytes 、diskio_writes 、diskio_reads Mem: mem_used_percent Net: net_bytes_sent 、net_bytes_recv 、net_packets_sent 、net_packets_recv Swap: swap_used_percent
スタンダード	CPU: cpu_usage_idle 、cpu_usage_iowait Disk: disk_used_percent 、disk_inodes_free Diskio: diskio_io_time 、diskio_write_bytes 、diskio_read_bytes 、diskio_writes 、diskio_reads Mem: mem_used_percent Net: net_bytes_sent 、net_bytes_recv 、net_packets_sent 、net_packets_recv Swap: swap_used_percent
アドバンスト	CPU: cpu_usage_guest 、cpu_usage_idle 、cpu_usage_iowait 、cpu_usage_steal 、cpu_usage_user 、cpu_usage_system Disk: disk_used_percent 、disk_inodes_free Diskio: diskio_io_time 、diskio_write_bytes 、diskio_read_bytes 、diskio_writes 、diskio_reads Mem: mem_used_percent Net: net_bytes_sent 、net_bytes_recv 、net_packets_sent 、net_packets_recv

詳細レベル	含められるメトリクス
	Netstat: `netstat_tcp_established` 、`netstat_tcp_time_wait` Swap: `swap_used_percent`

Windows Server を実行する Amazon EC2 インスタンス

> **Note**
>
> この表に記載されているメトリクス名は、メトリクスがどのようにコンソールに表示されるかを示しています。実際のメトリクス名には、最初の単語が含まれていない場合があります。例えば、`LogicalDisk % Free Space` の実際のメトリクス名は `% Free Space` です。

詳細レベル	含められるメトリクス
Basic (ベーシック)	Memory: `Memory % Committed Bytes In Use` LogicalDisk: `LogicalDisk % Free Space`
スタンダード	Memory: `Memory % Committed Bytes In Use` Paging: `Paging File % Usage` Processor: `Processor % Idle Time`、`Processor % Interrupt Time`、`Processor % User Time` PhysicalDisk: `PhysicalDisk % Disk Time` LogicalDisk: `LogicalDisk % Free Space`
アドバンスト	Memory: `Memory % Committed Bytes In Use` Paging: `Paging File % Usage` Processor: `Processor % Idle Time`、`Processor % Interrupt Time`、`Processor % User Time`

詳細レベル	含められるメトリクス
	LogicalDisk: `LogicalDisk % Free Space` PhysicalDisk: `PhysicalDisk % Disk Time` 、`PhysicalDisk Disk Write Bytes/sec` 、`PhysicalDisk Disk Read Bytes/sec` 、`PhysicalDisk Disk Writes/sec` 、`PhysicalDisk Disk Reads/sec` TCP: `TCPv4 Connections Established` 、 `TCPv6 Connections Established`

Windows Server を実行しているオンプレミスサーバー

> **Note**
>
> この表に記載されているメトリクス名は、メトリクスがどのようにコンソールに表示されるかを示しています。実際のメトリクス名には、最初の単語が含まれていない場合があります。例えば、`LogicalDisk % Free Space` の実際のメトリクス名は `% Free Space` です。

詳細レベル	含められるメトリクス
Basic (ベーシック)	Paging: `Paging File % Usage` Processor: `Processor % Processor Time` LogicalDisk:`LogicalDisk % Free Space` PhysicalDisk: `PhysicalDisk Disk Write Bytes/sec` 、`PhysicalDisk Disk Read Bytes/sec` 、`PhysicalDisk Disk Writes/sec` 、`PhysicalDisk Disk Reads/sec` Memory: `Memory % Committed Bytes In Use`

詳細レベル	含められるメトリクス
	Network Interface: Network Interface Bytes Sent/sec、Network Interface Bytes Received/sec 、Network Interface Packets Sent/sec、Network Interface Packets Received/sec
スタンダード	Paging: Paging File % Usage Processor: Processor % Processor Time、Processor % Idle Time、Processor % Interrupt Time LogicalDisk: LogicalDisk % Free Space PhysicalDisk: PhysicalDisk % Disk Time 、PhysicalDisk Disk Write Bytes/sec 、PhysicalDisk Disk Read Bytes/sec 、PhysicalDisk Disk Writes/sec 、PhysicalDisk Disk Reads/sec Memory: Memory % Committed Bytes In Use Network Interface: Network Interface Bytes Sent/sec、Network Interface Bytes Received/sec 、Network Interface Packets Sent/sec、Network Interface Packets Received/sec

詳細レベル	含められるメトリクス
アドバンスト	Paging:Paging File % Usage Processor: Processor % Processor Time、Processor % Idle Time、Processor % Interrupt Time、Processor % User Time LogicalDisk: LogicalDisk % Free Space PhysicalDisk: PhysicalDisk % Disk Time 、PhysicalDisk Disk Write Bytes/sec 、PhysicalDisk Disk Read Bytes/sec 、PhysicalDisk Disk Writes/sec 、PhysicalDisk Disk Reads/sec Memory: Memory % Committed Bytes In Use Network Interface: Network Interface Bytes Sent/sec、Network Interface Bytes Received/sec 、Network Interface Packets Sent/sec、Network Interface Packets Received/sec TCP: TCPv4 Connections Established 、 TCPv6 Connections Established

CloudWatch エージェント設定ファイルを手動で作成または編集する

CloudWatch エージェント設定ファイルは、agent、metrics、logs、traces の 4 つのセクションを持つ JSON ファイルです。各セクションは以下の通りです。

- agent セクションには、エージェントの全体的な設定に関するフィールドが含まれています。
- metrics セクションでは、収集と CloudWatch への発行に関するカスタムメトリクスを指定します。ログを収集するためだけにエージェントを使用している場合は、ファイルから metrics セクションを省略できます。
- logs セクションでは、CloudWatch Logs に発行されるログファイルを指定します。サーバーで Windows Server が実行されている場合、これには Windows イベントログからのイベントが含まれることがあります。
- traces セクションでは、収集されて AWS X-Ray に送信されるトレースのソースを指定します。

以下のセクションでは、この JSON ファイルの構造とフィールドについて説明します。この設定ファイルのスキーマ定義を表示することもできます。スキーマ定義は、Linux サーバーでは *installation-directory*/doc/amazon-cloudwatch-agent-schema.json、Windows Server を実行しているサーバーでは *installation-directory*/amazon-cloudwatch-agent-schema.json にあります。

エージェント設定ファイルを手動で作成または編集する場合は、任意の名前を付けることができます。トラブルシューティングを簡単にするため、Linux サーバーでは、/opt/aws/amazon-cloudwatch-agent/etc/amazon-cloudwatch-agent.json、Windows Server を実行しているサーバーでは、$Env:ProgramData\Amazon\AmazonCloudWatchAgent\amazon-cloudwatch-agent.json という名前を付けることをお勧めします。ファイルを作成したら、エージェントをインストールする他のサーバーにそのファイルをコピーできます。

> **Note**
>
> CloudWatch エージェントによって収集されたメトリクス、ログ、トレースには料金が発生します。料金の詳細については、「Amazon CloudWatch の料金」を参照してください。

CloudWatch エージェント設定ファイル: Agent セクション

agent セクションでは、次のフィールドを含めることができます。ウィザードでは、agent セクションは作成されません。代わりに、このセクションは省略されて、このセクションのすべてのフィールドにデフォルト値が使用されます。

- metrics_collection_interval - オプション。この設定ファイルで指定されたすべてのメトリクスが収集される頻度を指定します。この値は、特定の種類のメトリクスで上書きすることができます。

  この値は数秒で指定されます。例えば、10 と設定するとメトリクスが 10 秒ごとに収集されるようになり、300 と設定するとメトリクスが 5 分ごとに収集されように指定されます。

  この値を 60 秒未満に設定した場合、各メトリクスは高解像度メトリクスとして収集されます。高解像度メトリクスの詳細については、「高解像度のメトリクス」を参照してください。

  デフォルト値は 60 です。
- region – Amazon EC2 インスタンスを監視しているときに CloudWatch エンドポイントに使用するリージョンを指定します。収集されたメトリクスは、このリージョン (us-west-1 など) に送信

されます。このフィールドを省略すると、エージェントは Amazon EC2 インスタンスが配置されているリージョンにメトリクスを送信します。

オンプレミスサーバーをモニターリングしている場合、このフィールドは使用されず、エージェントは AmazonCloudWatchAgent 設定ファイルの AWS プロファイルからリージョンを読み取ります。

- credentials – 異なる AWS アカウントにメトリクス、ログ、トレースを送信する際に使用する IAM ロールを指定します。指定した場合、このフィールドには 1 つのパラメータ role_arn が含まれています。
  - role_arn – 異なる AWS アカウントにメトリクス、ログ、トレースを送信する際の認証用 IAM ロールの Amazon リソースネーム (ARN) を指定します。詳細については、「別のアカウントへのメトリクス、ログ、トレースの送信」を参照してください。
- debug - オプション。デバッグログメッセージを使用して CloudWatch を実行することを指定します。デフォルト値は false です。
- aws_sdk_log_level - オプション。バージョン 1.247350.0 以降の CloudWatch エージェントでのみサポートされます。

  このフィールドを指定すると、エージェントが AWS SDK エンドポイントに対してログ記録を実行できるようになります。このフィールドの値には、次のオプションうち 1 つまたは複数を含めることができます。複数のオプションは、| 文字で区切ります。
  - LogDebug
  - LogDebugWithSigning
  - LogDebugWithHTTPBody
  - LogDebugRequestRetries
  - LogDebugWithEventStreamBody

  これらのオプションの詳細については、「LogLevelType」を参照してください。
- logfile – CloudWatch エージェントがログメッセージを書き込む場所を指定します。空の文字列を指定すると、ログは stderr に書き込まれます。このオプションを指定しない場合、デフォルトの場所は次のようになります。
  - Linux: /opt/aws/amazon-cloudwatch-agent/logs/amazon-cloudwatch-agent.log
  - Windows Server: c:\\ProgramData\\Amazon\\CloudWatchAgent\\Logs\\amazon-cloudwatch-agent.log

CloudWatch エージェントで作成されるログファイルは自動的に更新されます。ログファイルは、サイズが 100 MB に達すると、更新されます。エージェントは、更新したログファイルを最大 7 日間保持します。また、更新したバックアップログファイルを最大 5 つまで保持します。バックアップログファイルのファイル名にはタイムスタンプが追加されます。このタイムスタンプは、ファイルの更新日時 (例: `amazon-cloudwatch-agent-2018-06-08T21-01-50.247.log.gz`) を示します。

- `omit_hostname` - オプション。デフォルトでは、`append_dimensions` セクション内の `metrics` フィールドを使用している場合を除き、ホスト名はエージェントによって収集されるメトリクスのディメンションとして発行されます。`omit_hostname` を `true` に設定すると、`append_dimensions` を使用していなくても、ホスト名がディメンションとして発行されません。デフォルト値は `false` です。
- `run_as_user` - オプション。CloudWatch エージェントを実行するために使用するユーザーを指定します。このパラメータを指定しないと、ルートユーザーが使用されます。このオプションは、Linux サーバーでのみ有効です。

  このオプションを使用する場合は、CloudWatch エージェントを起動する前にユーザーが存在している必要があります。詳細については、「CloudWatch エージェントの別のユーザーとしての実行」を参照してください。
- `user_agent` - オプション。CloudWatch エージェントによって CloudWatch バックエンドへの API 呼び出しに使用される `user-agent` 文字列を指定します。デフォルト値は、エージェントのバージョン、エージェントのコンパイルに使用された Go プログラミング言語のバージョン、ランタイムオペレーティングシステムおよびアーキテクチャ、構築時間、有効にするプラグインで構成される文字列です。
- `usage_data` - オプション。デフォルトでは、CloudWatch エージェントは、メトリクスまたはログを CloudWatch に発行するたびに、自らに関するヘルスおよびパフォーマンスのデータを CloudWatch に送信します。このデータには費用はかかりません。`usage_data` に `false` を指定することで、エージェントがこのデータを送信しないようにできます。このパラメータを省略すると、デフォルトの `true` が使用され、エージェントはヘルスおよびパフォーマンスのデータを送信します。

  この値を `false` に設定した場合、有効にするにはエージェントを停止して再起動する必要があります。

`agent` セクションの例を以下に示します。

```
"agent": {
   "metrics_collection_interval": 60,
   "region": "us-west-1",
   "logfile": "/opt/aws/amazon-cloudwatch-agent/logs/amazon-cloudwatch-agent.log",
   "debug": false,
   "run_as_user": "cwagent"
  }
```

CloudWatch エージェント設定ファイル: Metrics セクション

Linux および Windows の共通のフィールド

Linux または Windows Server を実行しているサーバーでは、metrics セクションに次のフィールドが含まれています。

- namespace - オプション。エージェントによって収集されるメトリクスに使用する名前空間。デフォルト値は CWAgent です。最大長は 255 文字です。次に例を示します。

  ```
  {
    "metrics": {
      "namespace": "Development/Product1Metrics",
     ......
     },
  }
  ```

- append_dimensions - オプション。エージェントによって収集されたメトリクスに Amazon EC2 メトリクスのディメンションを追加します。これにより、エージェントはホスト名をディメンションとして発行しません。

  append_dimensions で唯一サポートされるキーと値のペアは、次のリストに示されています。他のキーと値のペアは無視されます。エージェントは、次のリストに示されているとおりに、これらのキーと値のペアをサポートします。キーの値を変更して、異なるディメンション名を発行することはできません。

  - "ImageId":"${aws:ImageId}" は、インスタンスの AMI ID を ImageId ディメンションの値として設定します。
  - "InstanceId":"${aws:InstanceId}" は、インスタンスのインスタンス ID を InstanceId ディメンションの値として設定します。
  - "InstanceType":"${aws:InstanceType}" は、インスタンスのインスタンスタイプを InstanceType ディメンションの値として設定します。

- "AutoScalingGroupName":"${aws:AutoScalingGroupName}" は、インスタンスの Auto Scaling グループ名を AutoScalingGroupName ディメンションの値として設定します。

任意のキーと値のペアを持つメトリクスにディメンションを追加する場合は、その特定のタイプのメトリクスのフィールドの append_dimensions パラメータを使用します。

Amazon EC2 メタデータに依存する値を指定し、プロキシーを使用する場合は、サーバーが Amazon EC2 のエンドポイントにアクセスできることを確認する必要があります。これらのエンドポイントの詳細については、「Amazon Web Services 全般のリファレンス」の「Amazon Elastic Compute Cloud (Amazon EC2)」を参照してください。

- aggregation_dimensions - オプション。収集されたメトリクスが集計されるディメンションを指定します。例えば、AutoScalingGroupName ディメンションでメトリクスをロールアップした場合、各 Auto Scaling グループ内のすべてのインスタンスのメトリクスが集約され、全体として参照できるようになります。

  1 つまたは複数のディメンションに沿ってメトリクスをロールアップすることができます。たとえば、[["InstanceId"], ["InstanceType"], ["InstanceId","InstanceType"]] を指定すると、インスタンス ID 単独、インスタンスタイプ単独、および 2 つのディメンションの組み合わせのメトリクスが集約されます。

  [] を指定することにより、すべてのディメンションを無視して、すべてのメトリクスを 1 つのコレクションにロールアップすることもできます。

- endpoint_override – エージェントがメトリクスを送信するエンドポイントとして使用する FIPS エンドポイントまたはプライベートリンクを指定します。これを指定してプライベートリンクを設定すると、Amazon VPC エンドポイントにメトリクスを送信できます。詳細については、「Amazon VPC とは」を参照してください。

  endpoint_override の値は URL 文字列であることが必要です。

  たとえば、設定ファイルのメトリクスセクションの次の部分は、メトリクスの送信時に VPC エンドポイントを使用するようにエージェントを設定します。

```
{
  "metrics": {
    "endpoint_override": "vpce-XXXXXXXXXXXXXXXXXXXXXXXXX.monitoring.us-
east-1.vpce.amazonaws.com",
   ......
   },
```

```
}
```

- metrics_collected – 必須。収集するメトリクス (StatsD または collectd を介して収集されたカスタムメトリクスを含む) を指定します。このセクションには、いくつかのサブセクションがあります。

  この metrics_collected セクションの内容は、この設定ファイルが Linux を実行しているサーバー用か Windows Server を実行しているサーバー用かによって異なります。
- force_flush_interval – メトリクスがサーバーに送信されるまでにメモリバッファ内に残留する最大時間を秒単位で指定します。この設定にかかわらず、バッファ内のメトリクスのサイズが 1 MB に達するか、異なるタイプのメトリクスの数が 1,000 個に達すると、メトリクスは即座にサーバーに送信されます。

  デフォルト値は 60 です。
- credentials – 異なるアカウントにメトリクスを送信する際に使用する IAM ロールを指定します。指定した場合、このフィールドには 1 つのパラメータ role_arn が含まれています。
  - role_arn – 異なるアカウントにメトリクスを送信する際の認証用の IAM ロールの ARN を指定します。詳細については、「別のアカウントへのメトリクス、ログ、トレースの送信」を参照してください。ここで指定した値は、設定ファイルの role_arn セクションで指定された agent よりも優先されます (存在する場合)。

Linux のセクション

Linux を実行しているサーバーでは、設定ファイルの metrics_collected セクションに次のフィールドを含めることもできます。

これらのフィールドの多くには、そのリソースに対して収集するメトリクスをリストする measurement セクションを含めることができます。これらの measurement セクションでは、swap_used などの完全なメトリクス名、またはリソースのタイプに追加されるメトリクス名の一部のみを指定できます。たとえば、reads セクションの measurement セクションで diskio を指定すると、diskio_reads メトリクスが収集されます。

- collectd - オプション。collectd プロトコルを使用してカスタムメトリクスを取得することを指定します。collectd ソフトウェアを使用して CloudWatch エージェントにメトリクスを送信します。collectd で使用できる設定オプションの詳細については、「collectd を使用してカスタムメトリクスを取得する」を参照してください。
- cpu - オプション。収集する CPU メトリクスを指定します。このセクションは、Linux インスタンスでのみ有効です。収集する CPU メトリクスの resources フィールドと totalcpu フィール

ドを少なくとも 1 つ含める必要があります。このセクションには、次のフィールドを含めることができます。

- drop_original_metrics - オプション。metrics セクションの aggregation_dimensions フィールドを使用してメトリクスを集計結果にロールアップしている場合、デフォルトでは、エージェントは集計メトリクスと、ディメンションの値ごとに分離された元のメトリクスの両方を送信します。元のメトリクスを CloudWatch に送信したくない場合は、メトリクスのリストを使用してこのパラメータを指定できます。このパラメータとともに指定されたメトリクスには、CloudWatch にレポートされるディメンションごとのメトリクスがありません。代わりに、集計されたメトリクスのみがレポートされます。これにより、エージェントが収集するメトリクスの数が減り、コストが削減されます。
- resources - オプション。このフィールドの値を * に指定すると、CPU ごとのメトリクスが収集されます。許容されている値は * のみです。
- totalcpu - オプション。すべての CPU コア間で集計された cpu メトリクスを報告するかどうかを指定します。デフォルトは true です。
- measurement – 収集する cpu メトリクスの配列を指定します。利用できる値は、time_active、time_guest、time_guest_nice、time_idle、time_iowait、time_irq、および usage_user です。このフィールドは、cpu を含めた場合にのみ必要です。

  デフォルトでは、cpu_usage_* メトリクスの単位は Percent であり、cpu_time_* メトリクスには単位がありません。

  各メトリクスのエントリでは、必要に応じて次のいずれかまたは両方を指定できます。

  - rename – このメトリクスに別の名前を指定します。
  - unit – メトリクスの None のデフォルト単位を上書きして、このメトリクスに使用する単位を指定します。指定する単位は、「MetricDatum」の Unit に関する説明にリストされている有効な CloudWatch メトリクス単位でなければなりません。
- metrics_collection_interval - オプション。設定ファイルの metrics_collection_interval セクションで指定されたグローバルな agent を上書きして、cpu メトリクスを収集する頻度を指定します。

  この値は数秒で指定されます。例えば、10 と設定するとメトリクスが 10 秒ごとに収集されるようになり、300 と設定するとメトリクスが 5 分ごとに収集されように指定されます。

  この値を 60 秒未満に設定した場合、各メトリクスは高解像度メトリクスとして収集されます。高解像度メトリクスの詳細については、「高解像度のメトリクス」を参照してください。

- append_dimensions - オプション。cpu メトリクスにのみ使用する追加のディメンション。このフィールドを指定した場合、エージェントによって収集されるすべてのタイプのメトリクスに使用されるグローバルフィールド append_dimensions で指定されたディメンションに加えて使用されます。

- disk - オプション。収集する disk メトリクスを指定します。このセクションは、Linux インスタンスでのみ有効です。このセクションには、次のフィールドを含めることができます。

  - drop_original_metrics - オプション。metrics セクションの aggregation_dimensions フィールドを使用してメトリクスを集計結果にロールアップしている場合、デフォルトでは、エージェントは集計メトリクスと、ディメンションの値ごとに分離された元のメトリクスの両方を送信します。元のメトリクスを CloudWatch に送信したくない場合は、メトリクスのリストを使用してこのパラメータを指定できます。このパラメータとともに指定されたメトリクスには、CloudWatch にレポートされるディメンションごとのメトリクスがありません。代わりに、集計されたメトリクスのみがレポートされます。これにより、エージェントが収集するメトリクスの数が減り、コストが削減されます。

  - resources - オプション。ディスクのマウントポイントの配列を指定します。このフィールドは、リストされているマウントポイントのみから CloudWatch がメトリクスを収集するように制限します。値として * を指定すると、すべてのマウントポイントからメトリクスを収集できます。デフォルト値を使用すると、すべてのマウントポイントからメトリクスが収集されます。

  - measurement – 収集する disk メトリクスの配列を指定します。指定できる値は free、total、used、used_percent、inodes_free、inodes_used、および inodes_total です。このフィールドは、disk を含めた場合にのみ必要です。

    > **Note**
    >
    > disk メトリクスには Partition のディメンションがあります。つまり、生成されるカスタムメトリクスの数は、インスタンスに関連付けられたパーティションの数によって異なります。ディスクパーティションの数は、使用している AMI とサーバーにアタッチする Amazon EBS ボリュームの数によって異なります。

    各 disk メトリクスのデフォルトの単位を確認するには、「Linux および macOS インスタンスで CloudWatch エージェントにより収集されるメトリクス」を参照してください。

    各メトリクスのエントリでは、必要に応じて次のいずれかまたは両方を指定できます。

    - rename – このメトリクスに別の名前を指定します。

  - unit – メトリクスの None の None のデフォルト単位を上書きして、このメトリクスに使用する単位を指定します。指定する単位は、「MetricDatum」の Unit に関する説明にリストされている有効な CloudWatch メトリクス単位でなければなりません。
- ignore_file_system_types – disk メトリクスを収集するときに除外するファイルシステムのタイプを指定します。有効な値は、sysfs、devtmpfs などです。
- drop_device – これを true に設定すると、Device はディスクメトリクスのディメンションとして含まれません。

  Device がディメンションとして使用されないようにすることは、Nitro システムを使用するインスタンスで役立ちます。これらのインスタンスでは、インスタンスの再起動時にディスクマウントごとにデバイス名が変更されるためです。これにより、メトリクスに整合性のないデータが発生し、これらのメトリクスに基づくアラームが INSUFFICIENT DATA 状態になる可能性があります。

  デフォルト: false。
- metrics_collection_interval - オプション。設定ファイルの metrics_collection_interval セクションで指定されたグローバルな agent を上書きして、disk メトリクスを収集する頻度を指定します。

  この値は数秒で指定されます。

  この値を 60 秒未満に設定した場合、各メトリクスは高解像度メトリクスとして収集されます。詳細については、「高解像度のメトリクス」を参照してください。
- append_dimensions - オプション。disk メトリクスにのみ使用する追加のディメンション。このフィールドを指定した場合、エージェントにより収集されるすべてのタイプのメトリクスに使用される append_dimensions フィールドで指定されたディメンションに加えて使用されます。

- diskio - オプション。収集する disk i/o メトリクスを指定します。このセクションは、Linux インスタンスでのみ有効です。このセクションには、次のフィールドを含めることができます。
  - drop_original_metrics - オプション。metrics セクションの aggregation_dimensions フィールドを使用してメトリクスを集計結果にロールアップしている場合、デフォルトでは、エージェントは集計メトリクスと、ディメンションの値ごとに分離された元のメトリクスの両方を送信します。元のメトリクスを CloudWatch に送信したくない場合は、メトリクスのリストを使用してこのパラメータを指定できます。このパラメータとともに指定されたメトリクスには、CloudWatch にレポートされるディメンションごとのメトリク

スがありません。代わりに、集計されたメトリクスのみがレポートされます。これにより、エージェントが収集するメトリクスの数が減り、コストが削減されます。

- resources - オプション。デバイスの配列を指定した場合、CloudWatch はそれらのデバイスからのみメトリクスを収集します。それ以外の場合、すべてのデバイスのメトリクスが収集されます。値として * を指定し、すべてのデバイスからメトリクスを収集することもできます。
- measurement – 収集する diskio メトリクスの配列を指定します。指定できる値は reads、writes、read_bytes、write_bytes、read_time、write_time、io_time、および iops_in_progress です。このフィールドは、diskio を含めた場合にのみ必要です。

  各メトリクスのエントリでは、必要に応じて次のいずれかまたは両方を指定できます。

  - rename – このメトリクスに別の名前を指定します。
  - unit – メトリクスの None の None のデフォルト単位を上書きして、このメトリクスに使用する単位を指定します。指定する単位は、「MetricDatum」の Unit に関する説明にリストされている有効な CloudWatch メトリクス単位でなければなりません。
- metrics_collection_interval - オプション。設定ファイルの metrics_collection_interval セクションで指定されたグローバルな agent を上書きして、diskio メトリクスを収集する頻度を指定します。

  この値は数秒で指定されます。

  この値を 60 秒未満に設定した場合、各メトリクスは高解像度メトリクスとして収集されます。高解像度メトリクスの詳細については、「高解像度のメトリクス」を参照してください。
- append_dimensions - オプション。diskio メトリクスにのみ使用する追加のディメンション。このフィールドを指定した場合、エージェントにより収集されるすべてのタイプのメトリクスに使用される append_dimensions フィールドで指定されたディメンションに加えて使用されます。

- swap - オプション。収集するスワップメモリメトリクスを指定します。このセクションは、Linux インスタンスでのみ有効です。このセクションには、次のフィールドを含めることができます。
  - drop_original_metrics - オプション。metrics セクションの aggregation_dimensions フィールドを使用してメトリクスを集計結果にロールアップしている場合、デフォルトでは、エージェントは集計メトリクスと、ディメンションの値ごとに分離された元のメトリクスの両方を送信します。元のメトリクスを CloudWatch に送信したくない場合は、メトリクスのリストを使用してこのパラメータを指定できます。このパラメータとともに指定されたメトリクスには、CloudWatch にレポートされるディメンションごとのメトリクスがありません。代わりに、集計されたメトリクスのみがレポートされます。これにより、エージェントが収集するメトリクスの数が減り、コストが削減されます。

- measurement – 収集する swap メトリクスの配列を指定します。指定できる値は free、used、および used_percent です。このフィールドは、swap を含めた場合にのみ必要です。

  各 swap メトリクスのデフォルトの単位を確認するには、「Linux および macOS インスタンスで CloudWatch エージェントにより収集されるメトリクス」を参照してください。

  各メトリクスのエントリでは、必要に応じて次のいずれかまたは両方を指定できます。

  - rename – このメトリクスに別の名前を指定します。
  - unit – メトリクスの None の None のデフォルト単位を上書きして、このメトリクスに使用する単位を指定します。指定する単位は、「MetricDatum」の Unit に関する説明にリストされている有効な CloudWatch メトリクス単位でなければなりません。

- metrics_collection_interval - オプション。設定ファイルの metrics_collection_interval セクションで指定されたグローバルな agent を上書きして、swap メトリクスを収集する頻度を指定します。

  この値は数秒で指定されます。

  この値を 60 秒未満に設定した場合、各メトリクスは高解像度メトリクスとして収集されます。高解像度メトリクスの詳細については、「高解像度のメトリクス」を参照してください。

- append_dimensions - オプション。swap メトリクスにのみ使用する追加のディメンション。このフィールドを指定した場合、エージェントにより収集されるすべてのタイプのメトリクスに使用されるグローバル append_dimensions フィールドで指定されたディメンションに加えて使用されます。高解像度メトリクスとして収集されます。

- mem - オプション。収集するメモリメトリクスを指定します。このセクションは、Linux インスタンスでのみ有効です。このセクションには、次のフィールドを含めることができます。

  - drop_original_metrics - オプション。metrics セクションの aggregation_dimensions フィールドを使用してメトリクスを集計結果にロールアップしている場合、デフォルトでは、エージェントは集計メトリクスと、ディメンションの値ごとに分離された元のメトリクスの両方を送信します。元のメトリクスを CloudWatch に送信したくない場合は、メトリクスのリストを使用してこのパラメータを指定できます。このパラメータとともに指定されたメトリクスには、CloudWatch にレポートされるディメンションごとのメトリクスがありません。代わりに、集計されたメトリクスのみがレポートされます。これにより、エージェントが収集するメトリクスの数が減り、コストが削減されます。

- measurement – 収集する memory メトリクスの配列を指定します。指定できる値は active、available、available_percent、buffered、cached、free、inactive、total、u です。このフィールドは、mem を含めた場合にのみ必要です。

  各 mem メトリクスのデフォルトの単位を確認するには、「Linux および macOS インスタンスで CloudWatch エージェントにより収集されるメトリクス」を参照してください。

  各メトリクスのエントリでは、必要に応じて次のいずれかまたは両方を指定できます。

  - rename – このメトリクスに別の名前を指定します。
  - unit – メトリクスの None のデフォルト単位を上書きして、このメトリクスに使用する単位を指定します。指定する単位は、「MetricDatum」の Unit に関する説明にリストされている有効な CloudWatch メトリクス単位でなければなりません。

- metrics_collection_interval - オプション。設定ファイルの metrics_collection_interval セクションで指定されたグローバルな agent を上書きして、mem メトリクスを収集する頻度を指定します。

  この値は数秒で指定されます。

  この値を 60 秒未満に設定した場合、各メトリクスは高解像度メトリクスとして収集されます。高解像度メトリクスの詳細については、「高解像度のメトリクス」を参照してください。

- append_dimensions - オプション。mem メトリクスにのみ使用する追加のディメンション。このフィールドを指定した場合、エージェントによって収集されるすべてのタイプのメトリクスに使用される append_dimensions フィールドで指定されたディメンションに加えて使用されます。

- net - オプション。収集するネットワーキングメトリクスを指定します。このセクションは、Linux インスタンスでのみ有効です。このセクションには、次のフィールドを含めることができます。

  - drop_original_metrics - オプション。metrics セクションの aggregation_dimensions フィールドを使用してメトリクスを集計結果にロールアップしている場合、デフォルトでは、エージェントは集計メトリクスと、ディメンションの値ごとに分離された元のメトリクスの両方を送信します。元のメトリクスを CloudWatch に送信したくない場合は、メトリクスのリストを使用してこのパラメータを指定できます。このパラメータとともに指定されたメトリクスには、CloudWatch にレポートされるディメンションごとのメトリクスがありません。代わりに、集計されたメトリクスのみがレポートされます。これにより、エージェントが収集するメトリクスの数が減り、コストが削減されます。

- resources - オプション。ネットワークインターフェイスの配列を指定した場合、CloudWatch はそれらのインターフェイスからのみメトリクスを収集します。それ以外の場合、すべてのデバイスのメトリクスが収集されます。また、値として * を指定し、すべてのインターフェイスからメトリクスを収集することもできます。
- measurement – 収集する networking メトリクスの配列を指定します。指定できる値は bytes_sent、bytes_recv、drop_in、drop_out、err_in、err_out、packets_sent、および packets_recv です。このフィールドは、net を含めた場合にのみ必要です。

  各 net メトリクスのデフォルトの単位を確認するには、「Linux および macOS インスタンスで CloudWatch エージェントにより収集されるメトリクス」を参照してください。

  各メトリクスのエントリでは、必要に応じて次のいずれかまたは両方を指定できます。

  - rename – このメトリクスに別の名前を指定します。
  - unit – メトリクスの None のデフォルト単位を上書きして、このメトリクスに使用する単位を指定します。指定する単位は、「MetricDatum」の Unit に関する説明にリストされている有効な CloudWatch メトリクス単位でなければなりません。
- metrics_collection_interval - オプション。設定ファイルの metrics_collection_interval セクションで指定されたグローバルな agent を上書きして、net メトリクスを収集する頻度を指定します。

  この値は数秒で指定されます。例えば、10 と設定するとメトリクスが 10 秒ごとに収集されるようになり、300 と設定するとメトリクスが 5 分ごとに収集されように指定されます。

  この値を 60 秒未満に設定した場合、各メトリクスは高解像度メトリクスとして収集されます。高解像度メトリクスの詳細については、「高解像度のメトリクス」を参照してください。
- append_dimensions - オプション。net メトリクスにのみ使用する追加のディメンション。このフィールドを指定した場合、エージェントにより収集されるすべてのタイプのメトリクスに使用される append_dimensions フィールドで指定されたディメンションに加えて使用されます。

- netstat - オプション。収集する TCP 接続状態と UDP 接続メトリクスを指定します。このセクションは、Linux インスタンスでのみ有効です。このセクションには、次のフィールドを含めることができます。
  - drop_original_metrics - オプション。metrics セクション の aggregation_dimensions フィールドを使用してメトリクスを集計結果にロールアップしている場合、デフォルトでは、エージェントは集計メトリクスと、ディメンションの値ごとに分離された元のメトリクスの両方を送信します。元のメトリクスを CloudWatch に送信したくな

い場合は、メトリクスのリストを使用してこのパラメータを指定できます。このパラメータとともに指定されたメトリクスには、CloudWatch にレポートされるディメンションごとのメトリクスがありません。代わりに、集計されたメトリクスのみがレポートされます。これにより、エージェントが収集するメトリクスの数が減り、コストが削減されます。

- `measurement` – 収集する netstat メトリクスの配列を指定します。指定できる値は `tcp_close`、`tcp_close_wait`、`tcp_closing`、`tcp_established`、`tcp_fin_wait1`、`tcp_fi` および `udp_socket` です。このフィールドは、`netstat` を含めた場合にのみ必要です。

  各 `netstat` メトリクスのデフォルトの単位を確認するには、「Linux および macOS インスタンスで CloudWatch エージェントにより収集されるメトリクス」を参照してください。

  各メトリクスのエントリでは、必要に応じて次のいずれかまたは両方を指定できます。

  - `rename` – このメトリクスに別の名前を指定します。
  - `unit` – メトリクスの None のデフォルト単位を上書きして、このメトリクスに使用する単位を指定します。指定する単位は、「MetricDatum」の `Unit` に関する説明にリストされている有効な CloudWatch メトリクス単位でなければなりません。
- `metrics_collection_interval` - オプション。設定ファイルの `metrics_collection_interval` セクションで指定されたグローバルな `agent` を上書きして、netstat メトリクスを収集する頻度を指定します。

  この値は数秒で指定されます。

  この値を 60 秒未満に設定した場合、各メトリクスは高解像度メトリクスとして収集されます。高解像度メトリクスの詳細については、「高解像度のメトリクス」を参照してください。
- `append_dimensions` - オプション。netstat メトリクスにのみ使用する追加のディメンション。このフィールドを指定した場合、エージェントにより収集されるすべてのタイプのメトリクスに使用される `append_dimensions` フィールドで指定されたディメンションに加えて使用されます。

- `processes` - オプション。収集するプロセスメトリクスを指定します。このセクションは、Linux インスタンスでのみ有効です。このセクションには、次のフィールドを含めることができます。
  - `drop_original_metrics` - オプション。`metrics` セクションの `aggregation_dimensions` フィールドを使用してメトリクスを集計結果にロールアップしている場合、デフォルトでは、エージェントは集計メトリクスと、ディメンションの値ごとに分離された元のメトリクスの両方を送信します。元のメトリクスを CloudWatch に送信したくない場合は、メトリクスのリストを使用してこのパラメータを指定できます。このパラメータとともに指定されたメトリクスには、CloudWatch にレポートされるディメンションごとのメトリク

スがありません。代わりに、集計されたメトリクスのみがレポートされます。これにより、エージェントが収集するメトリクスの数が減り、コストが削減されます。

- measurement – 収集する processes メトリクスの配列を指定します。指定できる値は blocked、dead、idle、paging、running、sleeping、stopped、total、total_threads、w です。このフィールドは、processes を含めた場合にのみ必要です。

  すべての processes メトリクスで、デフォルトの単位は None です。

  各メトリクスのエントリでは、必要に応じて次のいずれかまたは両方を指定できます。

  - rename – このメトリクスに別の名前を指定します。
  - unit – メトリクスの None のデフォルト単位を上書きして、このメトリクスに使用する単位を指定します。指定する単位は、「MetricDatum」の Unit に関する説明にリストされている有効な CloudWatch メトリクス単位でなければなりません。
- metrics_collection_interval - オプション。設定ファイルの metrics_collection_interval セクションで指定されたグローバルな agent を上書きして、processes メトリクスを収集する頻度を指定します。

  この値は数秒で指定されます。例えば、10 と設定するとメトリクスが 10 秒ごとに収集されるようになり、300 と設定するとメトリクスが 5 分ごとに収集されように指定されます。

  この値を 60 秒未満に設定した場合、各メトリクスは高解像度メトリクスとして収集されます。詳細については、「高解像度のメトリクス」を参照してください。
- append_dimensions - オプション。process メトリクスにのみ使用する追加のディメンション。このフィールドを指定した場合、エージェントにより収集されるすべてのタイプのメトリクスに使用される append_dimensions フィールドで指定されたディメンションに加えて使用されます。

- nvidia_gpu - オプション。収集する NVIDIA GPU メトリクスを指定します。このセクションは、NVIDIA GPU アクセラレーターで設定され、NVIDIA System Management Interface (nvidia-smi) がインストールされているホスト上の Linux インスタンスについてのみ有効です。

  収集される NVIDIA GPU メトリクスには、他のアクセラレータータイプ用に収集されるメトリクスと区別するために、nvidia_smi_ という文字列のプレフィックスが付けられます。このセクションには、次のフィールドを含めることができます。

  - drop_original_metrics - オプション。metrics セクションの aggregation_dimensions フィールドを使用してメトリクスを集計結果にロールアップしている場合、デフォルトでは、エージェントは集計メトリクスと、ディメンションの値ごとに分離された元のメトリクスの両方を送信します。元のメトリクスを CloudWatch に送信したくな

い場合は、メトリクスのリストを使用してこのパラメータを指定できます。このパラメータとともに指定されたメトリクスには、CloudWatch にレポートされるディメンションごとのメトリクスがありません。代わりに、集計されたメトリクスのみがレポートされます。これにより、エージェントが収集するメトリクスの数が減り、コストが削減されます。

- `measurement` – 収集する NVIDIA GPU メトリクスの配列を指定します。ここで使用できる値のリストについては、NVIDIA GPU メトリクスを収集する の表の [Metric] (メトリクス) 列を参照してください。

  各メトリクスのエントリでは、必要に応じて次のいずれかまたは両方を指定できます。

  - `rename` – このメトリクスに別の名前を指定します。
  - `unit` – メトリクスの None のデフォルト単位を上書きして、このメトリクスに使用する単位を指定します。指定する単位は、「MetricDatum」の Unit に関する説明にリストされている有効な CloudWatch メトリクス単位でなければなりません。

- `metrics_collection_interval` - オプション。設定ファイルの `agent` セクションで指定されたグローバルな `metrics_collection_interval` を上書きして、NVIDIA GPU メトリクスを収集する頻度を指定します。

- `procstat` - オプション。個別のプロセスからメトリクスを取得することを指定します。procstat で使用できる設定オプションの詳細については、「procstat プラグインでプロセスメトリクスを収集する」を参照してください 。
- `statsd` - オプション。StatsD プロトコルを使用してカスタムメトリクスを取得することを指定します。CloudWatch エージェントは、プロトコルのデーモンとして機能します。スタンダード StatsD クライアントを使用して CloudWatch エージェントにメトリクスを送信します。StatsD で使用できる設定オプションの詳細については、「StatsD を使用してカスタムメトリクスを取得する 」を参照してください。
- `ethtool` - オプション。`ethtool` プラグインを使用してネットワークメトリクスを取得することを指定します。このプラグインは、標準 ethtool ユーティリティによって収集されたメトリクスと、Amazon EC2 インスタンスからのネットワークパフォーマンスメトリクスの両方をインポートできます。ethtool で使用できる設定オプションの詳細については、「ネットワークパフォーマンスメトリクスの収集」を参照してください。

Linux サーバーの `metrics` セクションの例を次に示します。この例では、3 つの CPU メトリクス、3 つの netstat メトリクス、3 つのプロセスメトリクス、および 1 つのディスクメトリクスが収集され、エージェントは `collectd` クライアントから追加のメトリクスを受信するように設定されています。

```
"metrics": {
    "aggregation_dimensions" : [["AutoScalingGroupName"], ["InstanceId",
 "InstanceType"],[]],
    "metrics_collected": {
      "collectd": {},
      "cpu": {
        "resources": [
          "*"
        ],
        "measurement": [
          {"name": "cpu_usage_idle", "rename": "CPU_USAGE_IDLE", "unit": "Percent"},
          {"name": "cpu_usage_nice", "unit": "Percent"},
          "cpu_usage_guest"
        ],
        "totalcpu": false,
        "drop_original_metrics": [ "cpu_usage_guest" ],
        "metrics_collection_interval": 10,
        "append_dimensions": {
          "test": "test1",
          "date": "2017-10-01"
        }
      },
      "netstat": {
        "measurement": [
          "tcp_established",
          "tcp_syn_sent",
          "tcp_close"
        ],
        "metrics_collection_interval": 60
      },
       "disk": {
        "measurement": [
          "used_percent"
        ],
        "resources": [
          "*"
        ],
        "drop_device": true
      },
      "processes": {
        "measurement": [
          "running",
          "sleeping",
```

```
          "dead"
        ]
      }
    },
    "append_dimensions": {
      "ImageId": "${aws:ImageId}",
      "InstanceId": "${aws:InstanceId}",
      "InstanceType": "${aws:InstanceType}",
      "AutoScalingGroupName": "${aws:AutoScalingGroupName}"
    }
  }
```

Windows Server

Windows Server の metrics_collected セクションでは、Windows パフォーマンスオブジェクトごとに Memory、Processor、LogicalDisk などのサブセクションを含めることができます。使用可能なオブジェクトとカウンターについては、Microsoft Windows のドキュメントの「パフォーマンスカウンター」を参照してください。

各オブジェクトのサブセクション内では、収集するカウンタの measurement 配列を指定します。measurement 配列は、設定ファイルで指定するオブジェクトごとに必要です。resources フィールドを指定して、メトリクスの収集元のインスタンスを指定することもできます。また、* に resources を指定し、すべてのインスタンスの別個のメトリクスを収集することもできます。インスタンスのあるカウンターの resources を省略すると、すべてのインスタンスのデータが 1 つのセットに集約されます。インスタンスのないカウンターの resources を省略すると、CloudWatch エージェントはカウンターを収集しません。カウンターにインスタンスがあるかどうかを判断するには、次のコマンドのいずれかを使用します。

Powershell:

```
Get-Counter -ListSet *
```

コマンドライン (Powershell ではない):

```
TypePerf.exe -q
```

各オブジェクトセクション内で、以下のオプションフィールドを指定することもできます。

- metrics_collection_interval - オプション。設定ファイルの metrics_collection_interval セクションで指定されたグローバルな agent を上書きして、このオブジェクトのメトリクスを収集する頻度を指定します。

この値は数秒で指定されます。例えば、10 と設定するとメトリクスが 10 秒ごとに収集されるようになり、300 と設定するとメトリクスが 5 分ごとに収集されように指定されます。

この値を 60 秒未満に設定した場合、各メトリクスは高解像度メトリクスとして収集されます。詳細については、「高解像度のメトリクス」を参照してください。

- append_dimensions - オプション。このオブジェクトのメトリクスにのみ使用する追加のディメンションを指定します。このフィールドを指定した場合、エージェントにより収集されるすべてのタイプのメトリクスに使用されるグローバル append_dimensions フィールドで指定されたディメンションに加えて使用されます。
- drop_original_metrics - オプション。metrics セクションの aggregation_dimensions フィールドを使用してメトリクスを集計結果にロールアップしている場合、デフォルトでは、エージェントは集計メトリクスと、ディメンションの値ごとに分離された元のメトリクスの両方を送信します。元のメトリクスを CloudWatch に送信したくない場合は、メトリクスのリストを使用してこのパラメータを指定できます。このパラメータとともに指定されたメトリクスには、CloudWatch にレポートされるディメンションごとのメトリクスがありません。代わりに、集計されたメトリクスのみがレポートされます。これにより、エージェントが収集するメトリクスの数が減り、コストが削減されます。

各カウンタセクション内で、以下のオプションフィールドを指定することもできます。

- rename – このメトリクスに、CloudWatch で使用する別の名前を指定します。
- unit – このメトリクスに使用する単位を指定します。指定する単位は、「MetricDatum」の Unit に関する説明にリストされている有効な CloudWatch メトリクス単位でなければなりません。

metrics_collected には、他の 2 つのオプションのセクションを含めることができます。

- statsd – StatsD プロトコルを使用してカスタムメトリクスを取得できます。CloudWatch エージェントは、プロトコルのデーモンとして機能します。スタンダード StatsD クライアントを使用して CloudWatch エージェントにメトリクスを送信します。詳細については、「StatsD を使用してカスタムメトリクスを取得する 」を参照してください。
- procstat – 個別のプロセスからメトリクスを取得できます。詳細については、「procstat プラグインでプロセスメトリクスを収集する」を参照してください。

Windows Server で使用する metrics セクションの例を次に示します。この例では、多くの Windows メトリクスが収集され、コンピュータは StatsD クライアントから追加のメトリクスを受信するように設定されています。

```
"metrics": {
    "metrics_collected": {
      "statsd": {},
      "Processor": {
        "measurement": [
          {"name": "% Idle Time", "rename": "CPU_IDLE", "unit": "Percent"},
          "% Interrupt Time",
          "% User Time",
          "% Processor Time"
        ],
        "resources": [
          "*"
        ],
        "append_dimensions": {
          "d1": "win_foo",
          "d2": "win_bar"
        }
      },
      "LogicalDisk": {
        "measurement": [
          {"name": "% Idle Time", "unit": "Percent"},
          {"name": "% Disk Read Time", "rename": "DISK_READ"},
          "% Disk Write Time"
        ],
        "resources": [
          "*"
        ]
      },
      "Memory": {
        "metrics_collection_interval": 5,
        "measurement": [
          "Available Bytes",
          "Cache Faults/sec",
          "Page Faults/sec",
          "Pages/sec"
        ],
        "append_dimensions": {
          "d3": "win_bo"
        }
```

```
      },
      "Network Interface": {
        "metrics_collection_interval": 5,
        "measurement": [
          "Bytes Received/sec",
          "Bytes Sent/sec",
          "Packets Received/sec",
          "Packets Sent/sec"
        ],
        "resources": [
          "*"
        ],
        "append_dimensions": {
          "d3": "win_bo"
        }
      },
      "System": {
        "measurement": [
          "Context Switches/sec",
          "System Calls/sec",
          "Processor Queue Length"
        ],
        "append_dimensions": {
          "d1": "win_foo",
          "d2": "win_bar"
        }
      }
    },
    "append_dimensions": {
      "ImageId": "${aws:ImageId}",
      "InstanceId": "${aws:InstanceId}",
      "InstanceType": "${aws:InstanceType}",
      "AutoScalingGroupName": "${aws:AutoScalingGroupName}"
    },
    "aggregation_dimensions" : [["ImageId"], ["InstanceId", "InstanceType"], ["d1"],[]]
    }
  }
```

CloudWatch エージェント設定ファイル: Logs セクション

logs セクションには、次のフィールドが含まれます。

- logs_collected – logs セクションが含まれている場合に必要です。サーバーから収集するログファイルおよび Windows イベントログを指定します。files と windows_events の 2 つのフィールドを含めることができます。
  - files – CloudWatch エージェントが収集する定期的なログファイルを指定します。1 つのフィールド collect_list が含まれます。このフィールドでは、これらのファイルを詳細に定義します。
    - collect_list – files が含まれている場合に必要です。エントリの配列が含まれており、それぞれ収集するログファイルを 1 つ指定します。これらの各エントリには、次のフィールドを含めることができます。
      - file_path – CloudWatch Logs にアップロードするログファイルのパスを指定します。スーパーアスタリスクとして ** を追加すると、標準 Unix glob マッチングルールが受け入れられます。たとえば、/var/log/**.log と指定すると .log ディレクトリツリー内のすべての /var/log ファイルが収集されます。他の例については、「Glob Library」を参照してください。

        また、標準のアスタリスクを標準のワイルドカードとして使用することもできます。例えば、/var/log/system.log* は、/var/log 内の、system.log_1111、system.log_2222 などのファイルに一致します。

        ファイルの変更時間に基づいて、最新のファイルのみが CloudWatch Logs にプッシュされます。access_log.2018-06-01-01 と access_log.2018-06-01-02 など同じ形式の一連のファイルを指定するにはワイルドカードの使用をお勧めします。ただし、access_log_80 と access_log_443 のように複数の種類のファイルには使用しないでください。複数の種類のファイルを指定するには、エージェント設定ファイルに別のストリームログのエントリを追加して、各種類のログファイルが異なるログストリームに行くようにします。
      - auto_removal - オプション。これが true である場合、CloudWatch エージェントは、このログファイルを、読み取りとローテーションが完了した後に自動的に削除します。通常、ログファイルは、そのコンテンツ全体が CloudWatch Logs にアップロードされた後に削除されますが、エージェントが EOF (ファイルの終わり) に達し、同じ file_path に一致する別の新しいログファイルも検出した場合、エージェントは古いファイルを削除します。したがって、新しいファイルの作成前に、古いファイルへの記述を確実に完了する必要があります。RUST トレースライブラリには既知の非互換性があります。これは、新しいログファイルが作成された後も、古いログファイルへの書き込みが試行される可能性があることが原因です。

エージェントは、日別に異なるファイルを作成するログなど、複数のファイルを作成するログから完全なファイルのみを削除します。ログが 1 つのファイルに継続的に書き込まれる場合、そのファイルは削除されません。

ログファイルの更新方法や削除方法が設定済みである場合は、このフィールドを省略するか、false に設定することをお勧めします。

このフィールドを省略した場合は、デフォルト値の false が使用されます。

- log_group_name - オプション。CloudWatch Logs でロググループ名として何を使用するかを指定します。

  混乱を避けるため、このフィールドを使用してロググループ名を指定することをお勧めします。log_group_name を省略した場合、末尾のドットまでの file_path の値がロググループ名として使用されます。たとえば、ファイルパスが /tmp/TestLogFile.log.2017-07-11-14 の場合、ロググループ名は /tmp/TestLogFile.log です。

  ロググループ名を指定する場合、{instance_id}、{hostname}、{local_hostname}、{ip_address} を変数として使用できます。{hostname} は EC2 メタデータからホスト名を取得し、{local_hostname} はネットワーク設定ファイルからホスト名を使用します。

  これらの変数を使用して多くの異なるロググループを作成する場合は、1 リージョン、1 アカウントあたり 100 万ロググループという制限を念頭に置いてください。

  ここで使えるのは、a～z、A～Z、0～9、"_" (アンダーバー)、"-" (ハイフン)、"/" (スラッシュ) および "." (ピリオド) です。

- log_group_class - オプション。新しいロググループに使用するロググループクラスを指定します。ロググループクラスの詳細については、「Log classes」を参照してください。

  有効な値は、STANDARD および INFREQUENT_ACCESS です。このフィールドを省略した場合、STANDARD のデフォルトが使用されます。

> **Important**
>
> ロググループの作成後に、ロググループクラスを変更することはできません。

- log_stream_name - オプション。CloudWatch Logs でログストリーム名として何を使用するかを指定します。名前の一部として、名前では変数として {instance_id}、{hostname}、{local_hostname}、{ip_address} を使用することができます。{hostname} は、EC2 メタデータからホスト名を取得し、{local_hostname} は、ネットワーク設定ファイルからホスト名を使用します。

  このフィールドを省略すると、グローバル log_stream_name セクションの logs パラメータの値が使用されます。これも省略すると、{instance_id} のデフォルト値が使用されます。

  ログストリームが存在しない場合には、自動的に作成されます。

- retention_in_days - オプション。指定されたロググループにログイベントを保持する日数を指定します。
  - エージェントがこのロググループを作成中なら、このフィールドを省略すると、この新しいロググループが無期限に保持されるように設定されます。
  - このロググループが既に存在していて、このフィールドを指定すると、指定した新しい保存期間が使用されます。既に存在するロググループで、このフィールドを省略した場合、ロググループの保持は変更されません。

    CloudWatch エージェントウィザードは、このフィールドのデフォルト値として -1 を使用します。これは、エージェント設定ファイルを作成する場合で、ログ保持の値を指定しない場合に使用されます。ウィザードが設定するこの -1 の値によって、ロググループ内のイベントが期限切れにならないよう指定されます。ただし、この値を手動で -1 に編集しても効果はありません。

  有効な値は
  1、3、5、7、14、30、60、90、120、150、180、365、400、545、731、1827、2192、2557、29
  および 3653 です。

  同じロググループに複数のログストリームを書き込むようにエージェントを設定する場合、1 つの場所で retention_in_days を指定すると、ロググループ全体のログの保持が設定されます。複数の場所で同じロググループ向けに retention_in_days を指定した場合、それらの値がすべて等しい場合に保持が設定されます。ただし、複数の場所で同じロググループに異なる retention_in_days の値が指定されている場合、ログの保持は設定されず、エージェントは停止してエラーを返します。

> **Note**
>
> エージェントの IAM ロールまたは IAM ユーザーが保持ポリシーを設定するには、`logs:PutRetentionPolicy` を持っている必要があります。詳細については、「CloudWatch エージェントによるログの保持ポリシーの設定を許可」を参照してください。

> **Warning**
>
> 既に存在するロググループ向けに `retention_in_days` を設定した場合、指定した日数より前に発行されたそのロググループ内のすべてのログが削除されます。例えば、3 に設定すると、3 日以上前のすべてのログが削除されます。

- `filters` - オプション。エントリの配列を含めることができます。各エントリは、正規表現とフィルタータイプを指定し、フィルターに一致するログエントリを発行またはドロップするかどうかを指定します。このフィールドを省略すると、ログファイル内のすべてのログが CloudWatch Logs に発行されます。このフィールドを含めると、エージェントは指定したすべてのフィルターを使用して各ログメッセージを処理し、すべてのフィルターを通過するログイベントのみが CloudWatch Logs に発行されます。一部のフィルターを通過しないログエントリは、ホストのログファイルに残りますが、CloudWatch Logs には送信されません。

  フィルター配列の各エントリには、次のフィールドを含めることができます。

  - `type` – フィルターのタイプを示します。有効な値は、`include` および `exclude` です。`include` では、ログエントリは CloudWatch Logs に発行される式と一致する必要があります。`exclude` では、フィルターに一致する各ログエントリは CloudWatch Logs に送信されません。
  - `expression` – RE2 Syntax に続く正規表現の文字列。

    > **Note**
    >
    > CloudWatch エージェントは、指定した正規表現のパフォーマンスをチェックしたり、正規表現の評価の実行時間を制限したりしません。評価に多くのコストがかかる表現を記述しないように注意することをおすすめします。発生する

可能性のある問題の詳細については、「Regular expression Denial of Service - ReDoS」(正規表現のサービス拒否 - ReDoS) を参照してください。

例えば、次の CloudWatch エージェント設定ファイルの抜粋は、PUT および POST リクエストである ログを CloudWatch Logs に発行しますが、Firefox からのログは除外されます。

```
"collect_list": [
  {
    "file_path": "/opt/aws/amazon-cloudwatch-agent/logs/test.log",
    "log_group_name": "test.log",
    "log_stream_name": "test.log",
    "filters": [
      {
        "type": "exclude",
        "expression": "Firefox"
      },
      {
        "type": "include",
        "expression": "P(UT|OST)"
      }
    ]
  },
  .....
]
```

> **Note**
>
> 設定ファイル内のフィルターの順序は、パフォーマンス上重要です。前述の例において、エージェントは 2 番目のフィルターの評価を開始する前に、Firefox に一致するすべてのログをドロップします。複数のフィルターで評価されるログエントリを少なくするには、より多くのログを除外するフィルターを設定ファイルに最初に配置します。

- `timezone` - オプション。ログイベントにタイムスタンプを付けるときに使用するタイムゾーンを指定します。有効な値は `UTC` および `Local` です。デフォルト値は `Local` です。

  `timestamp_format` の値を指定しない場合、このパラメータは無視されます。

- `timestamp_format` - オプション。% で始まる特殊記号とプレーンテキストを使用して、タイムスタンプ形式を指定します。このフィールドを省略した場合は、現在の時刻が使用さ

れます。このフィールドを使用する場合は、以下のリストの記号を形式の一部に使用することができます。

1 つのログエントリに、形式に一致する 2 つのタイムスタンプが含まれている場合、最初のタイムスタンプが使用されます。

この記号のリストは、以前の CloudWatch Logs エージェントで使用されるリストとは異なります。このような違いの概要については、「統合 CloudWatch エージェントと前の CloudWatch Logs エージェントの間のタイムスタンプの違い」を参照してください。

%y

ゼロ詰め 10 進数での年 (世紀なし) たとえば 19 で 2019 を表します。

%Y

10 進数での年 (世紀あり) 例えば、2019 と指定します。

%b

ロケールの省略名称での月

%B

ロケールの正式名称での月

%m

ゼロ詰め 10 進数での月

%-m

10 進数での月 (ゼロ詰めなし)

%d

ゼロ詰め 10 進数での日

%-d

10 進数での日 (ゼロ詰めなし)

%A

曜日のフルネーム。Monday など

`%a`

曜日の略称。Mon など

`%H`

ゼロ詰め 10 進数での時 (24 時間形式の時計)

`%I`

ゼロ詰め 10 進数での時 (12 時間形式の時計)

`%-I`

10 進数での時間 (12 時間制) (ゼロ詰めなし)

`%p`

AM または PM

`%M`

ゼロ詰め 10 進数での分

`%-M`

10 進数での分 (ゼロ詰めなし)

`%S`

ゼロ詰め 10 進数での秒

`%-S`

10 進数での秒 (ゼロ詰めなし)

`%f`

10 進数 (1 ~ 9 桁) での少数秒 (左側をゼロ詰め)

`%Z`

タイムゾーン。PST など

`%z`

タイムゾーンは、ローカルタイムゾーンと UTC の間のオフセットとして表されます。例えば、`-0700` と指定します。この形式のみがサポートされています。たとえば、`-07:00` は有効な形式ではありません。

- multi_line_start_pattern – ログメッセージの開始を識別するパターンを指定します。ログメッセージは、パターンに一致する 1 行と、後続パターンに一致しない行で構成されます。

  このフィールドを指定しない場合、複数行モードが無効になり、空白文字以外の文字で始まる行は、前のログメッセージを終了して新しいログメッセージを開始します。

  このフィールドを含める場合、{timestamp_format} を指定してタイムスタンプ形式と同じ正規表現を使用できます。それ以外の場合、CloudWatch Logs が複数行エントリの最初の行を判断するために使用する別の正規表現を指定できます。

- encoding – 正しく読み込むことができるように、ログファイルのエンコードを指定します。正しくないエンコードを指定すると、デコードできない文字がその他の文字に置き換えられるため、データ損失が生じる可能性があります。

  デフォルト値は utf-8 です。指定できる値は以下のとおりです。

  ascii, big5, euc-jp, euc-kr, gbk, gb18030, ibm866, iso2022-jp, iso8859-2, iso8859-3, iso8859-4, iso8859-5, iso8859-6, iso8859-7, iso8859-8, iso8859-8-i, iso8859-10, iso8859-13, iso8859-14, iso8859-15, iso8859-16, koi8-r, koi8-u, macintosh, shift_jis, utf-8, utf-16, utf-16le, UTF-16, UTF-16LE, windows-874, windows-1250, windows-1251, windows-1252, windows-1253, windows-1254, windows-1255, windows-1256, windows-1257, windows-1258, x-mac-cyrillic

- windows_events セクションでは、Windows Server を実行しているサーバーから収集する Windows イベントのタイプを指定します。次のフィールドが含まれています。
  - collect_list – windows_events が含まれている場合に必要です。収集する Windows イベントのタイプとレベルを指定します。収集される各ログには、このセクションのエントリがあります。次のフィールドが含まれる可能性があります。
    - event_name – ログ記録する Windows イベントのタイプを指定します。これは Windows イベントログのチャネル名と同等です。たとえば、System、Security、Application などがあります。このフィールドは、ログ記録する Windows イベントのタイプごとに必要です。

> **Note**
>
> CloudWatch が Windows ログチャネルからメッセージを取得する際には、Full Name プロパティに基づき、そのログチャネルがルックアップされます。その間、Windows イベントビューアのナビゲーションペインには、ログチャネルの Log Name プロパティが表示されます。Full Name と Log Name は必ずしも一致しません。チャンネルの Full Name を確認するには、Windows イベントビューアでチャンネルを右クリックし、[プロパティ] を開きます。

- event_levels – ログ記録するイベントのレベルを指定します。ログ記録する各レベルを指定する必要があります。指定できる値には、INFORMATION、WARNING、ERROR、CRITICALおよび VERBOSE などがあります。このフィールドは、ログ記録する Windows イベントのタイプごとに必要です。
- log_group_name – 必須。CloudWatch Logs でロググループ名として何を使用するかを指定します。
- log_stream_name - オプション。CloudWatch Logs でログストリーム名として何を使用するかを指定します。名前の一部として、名前では変数として {instance_id}、{hostname}、{local_hostname}、{ip_address} を使用することができます。{hostname} は、EC2 メタデータからホスト名を取得し、{local_hostname} は、ネットワーク設定ファイルからホスト名を使用します。

  このフィールドを省略すると、グローバル log_stream_name セクションの logs パラメータの値が使用されます。これも省略すると、{instance_id} のデフォルト値が使用されます。

  ログストリームが存在しない場合には、自動的に作成されます。
- event_format - オプション。Windows のイベントを CloudWatch Logs に保存する際に使用する形式を指定します。xml は、Windows イベントビューワーと同じように XML 形式を使用します。text は、レガシーの CloudWatch Logs エージェント形式を使用します。
- retention_in_days - オプション。指定されたロググループに Windows イベントを保持する日数を指定します。
  - エージェントがこのロググループを作成中なら、このフィールドを省略すると、この新しいロググループが無期限に保持されるように設定されます。

- このロググループが既に存在していて、このフィールドを指定すると、指定した新しい保存期間が使用されます。既に存在するロググループで、このフィールドを省略した場合、ロググループの保持は変更されません。

  CloudWatch エージェントウィザードは、このフィールドのデフォルト値として -1 を使用します。これは、エージェント設定ファイルを作成する場合で、ログ保持の値を指定しない場合に使用されます。ウィザードが設定するこの -1 の値は、ロググループ内のイベントが期限切れにならないよう指定します。ただし、この値を手動で -1 に編集しても効果はありません。

  有効な値は
  1、3、5、7、14、30、60、90、120、150、180、365、400、545、731、1827、2192、2557、29
  および 3653 です。

  同じロググループに複数のログストリームを書き込むようにエージェントを設定する場合、1 つの場所で retention_in_days を指定すると、ロググループ全体のログの保持が設定されます。複数の場所で同じロググループ向けに retention_in_days を指定した場合、それらの値がすべて等しい場合に保持が設定されます。ただし、複数の場所で同じロググループに異なる retention_in_days の値が指定されている場合、ログの保持は設定されず、エージェントは停止してエラーを返します。

  > **Note**
  >
  > エージェントの IAM ロールまたは IAM ユーザーが保持ポリシーを設定するには、logs:PutRetentionPolicy を持っている必要があります。詳細については、「CloudWatch エージェントによるログの保持ポリシーの設定を許可」を参照してください。

  > **Warning**
  >
  > 既に存在するロググループ向けに retention_in_days を設定した場合、指定した日数より前に発行されたそのロググループ内のすべてのログが削除されます。例えば、3 に設定すると、3 日以上前のすべてのログが削除されます。

- log_stream_name – 必須。log_stream_name のログや Windows イベントの個別のログストリーム名が collect_list パラメータに定義されていない場合、代わりに使用するデフォルトのログストリーム名を指定します。

- endpoint_override – エージェントがログを送信するエンドポイントとして使用する FIPS エンドポイントまたはプライベートリンクを指定します。このフィールドを指定してプライベートリンクを設定すると、Amazon VPC エンドポイントにログを送信できます。詳細については、「Amazon VPC とは」を参照してください。

  endpoint_override の値は URL 文字列であることが必要です。

  たとえば、設定ファイルの logs セクションの次の部分は、ログを送信するときに VPC エンドポイントを使用するようにエージェントを設定します。

```
{
  "logs": {
    "endpoint_override": "vpce-XXXXXXXXXXXXXXXXXXXXXXXXX.logs.us-
east-1.vpce.amazonaws.com",
   ......
   },
}
```

- force_flush_interval – ログがサーバーに送信されるまでにメモリバッファ内に残留する最大時間を秒単位で指定します。このフィールドの設定にかかわらず、バッファ内のログのサイズが 1 MB に達すると、ログは即座にサーバーに送信されます。デフォルト値は 5 です。

  エージェントを使用して高解像度メトリクスを埋め込みメトリクス形式で報告し、それらのメトリクスにアラームを設定する場合は、このパラメータをデフォルト値の 5 に設定してください。それ以外の場合は、メトリクスが遅延して報告され、データが部分的または不完全な場合にアラームが発報する可能性があります。

- credentials – 異なる AWS アカウントにログを送信する際に使用する IAM ロールを指定します。指定した場合、このフィールドには 1 つのパラメータ role_arn が含まれています。
  - role_arn – 異なる AWS アカウントにログを送信する際の認証用 IAM ロールの ARN を指定します。詳細については、「別のアカウントへのメトリクス、ログ、トレースの送信」を参照してください。ここで指定した場合は、設定ファイルの role_arn セクションで指定された agent よりも優先されます (存在する場合)。
- metrics_collected - このフィールドには、エージェントにログ収集を指示するセクションを含めることができます。CloudWatch Application Signals と Container Insights のオブザーバビリティを Amazon EKS 向けに強化するといったユースケースを利用できるようになります。
  - app_signals(オプション) CloudWatch Application Signals を有効にすることを指定します。この設定の詳細については、「CloudWatch Application Signals を有効にする」を参照してください。

- kubernetes — このフィールドには、enhanced_container_insights パラメータを含めることができます。これは、Amazon EKS 向けにオブザーバビリティが強化された Container Insights を有効にするために使用できます。
  - enhanced_container_insights — これを true に設定すると、Amazon EKS 向けにオブザーバビリティが強化された Container Insights が有効になります。詳細については、「Amazon EKS 向けにオブザーバビリティが強化された Container Insights」を参照してください。
  - accelerated_compute_metrics - Amazon EKS クラスターでの Nvidia GPU メトリクスの収集をオプトアウトするには、これを false に設定します。詳細については、「NVIDIA GPU メトリクス」を参照してください。
- emf – ログに埋め込まれたメトリクスを収集するために、この emf フィールドを追加する必要はなくなりました。これは、エージェントが埋め込みメトリクス形式のログを収集するように指定するレガシーフィールドです。これらのログからメトリクスデータを生成できます。詳細については、「ログ内へのメトリクスの埋め込み」を参照してください。

logs セクションの例を以下に示します。

```
"logs":
   {
       "logs_collected": {
           "files": {
               "collect_list": [
                   {
                       "file_path": "c:\\ProgramData\\Amazon\\AmazonCloudWatchAgent\
\Logs\\amazon-cloudwatch-agent.log",
                       "log_group_name": "amazon-cloudwatch-agent.log",
                       "log_stream_name": "my_log_stream_name_1",
                       "timestamp_format": "%H: %M: %S%y%b%-d"
                   },
                   {
                       "file_path": "c:\\ProgramData\\Amazon\\AmazonCloudWatchAgent\
\Logs\\test.log",
                       "log_group_name": "test.log",
                       "log_stream_name": "my_log_stream_name_2"
                   }
               ]
           },
           "windows_events": {
               "collect_list": [
```

```
                {
                    "event_name": "System",
                    "event_levels": [
                        "INFORMATION",
                        "ERROR"
                    ],
                    "log_group_name": "System",
                    "log_stream_name": "System"
                },
                {
                    "event_name": "CustomizedName",
                    "event_levels": [
                        "INFORMATION",
                        "ERROR"
                    ],
                    "log_group_name": "CustomizedLogGroup",
                    "log_stream_name": "CustomizedLogStream"
                }
            ]
        }
     },
     "log_stream_name": "my_log_stream_name",
     "metrics_collected": {
         "kubernetes": {
              "enhanced_container_insights": true
    }
  }
}
```

CloudWatch エージェント設定ファイル: [トレース] セクション

CloudWatch エージェント設定ファイルに `traces` セクションを追加することで、CloudWatch Application Signals を有効にできるほか、X-Ray や OpenTelemetry 計測 SDK からトレースを収集して X-Ray に送信することもできます。

> **Important**
>
> エージェントの IAM ロールまたは IAM ユーザーがトレースデータを X-Ray に送信できるようにするには、AWSXrayWriteOnlyAccess ポリシーが必要です。詳細については、「CloudWatch エージェントで使用する IAM ロールとユーザーを作成する」を参照してください。

トレースの収集を簡単に開始するために必要なのは、CloudWatch エージェント設定ファイルに次の内容を追加することだけです。

```
"traces_collected": {
        "xray": {
        },
        "otlp": {
        }
      }
```

前のセクションを CloudWatch エージェント設定ファイルに追加してエージェントを再起動すると、エージェントは次のデフォルトのオプションと値を使用してトレースの収集を開始します。これらのパラメータの詳細については、このセクションで後述するパラメータの定義を参照してください。

```
"traces_collected": {
        "xray": {
            "bind_address": "127.0.0.1:2000",
            "tcp_proxy": {
              "bind_address": "127.0.0.1:2000"
            }
        },
        "otlp": {
            "grpc_endpoint": "127.0.0.1:4317",
            "http_endpoint": "127.0.0.1:4318"
        }
      }
```

`traces` セクションでは、次のフィールドを含めることができます。

- `traces_collected` – `traces` セクションが含まれている場合に必要です。トレースの収集元になる SDK を指定します。次のフィールドを含めることができます。
  - `app_signals` - オプション。CloudWatch Application Signals を有効にすることを指定します。この設定の詳細については、「CloudWatch Application Signals を有効にする」を参照してください。
  - `xray` - オプション。X-Ray SDK からトレースを収集することを指定します。このセクションには、次のフィールドを含めることができます。
    - `bind_address` - オプション。CloudWatch エージェントが X-Ray のトレースのリッスンに使用する UDP アドレスを指定します。形式は `ip:port` です。このアドレスは、X-Ray SDK で設定されたアドレスと一致する必要があります。

このフィールドを省略した場合、127.0.0.1:2000 のデフォルトが使用されます。

- tcp_proxy - オプション。X-Ray リモートサンプリングをサポートするために使用されるプロキシのアドレスを設定します。詳細については、X-Ray ドキュメントの「サンプリングルールの設定」を参照してください。

  このセクションには、次のフィールドを含めることができます。

  - bind_address - オプション。CloudWatch エージェントがプロキシを設定する必要がある TCP アドレスを指定します。形式は ip:port です。このアドレスは、X-Ray SDK で設定されたアドレスと一致する必要があります。

    このフィールドを省略した場合、127.0.0.1:2000 のデフォルトが使用されます。

- otlp - オプション。OpenTelemetry SDK からトレースを収集することを指定します。AWS Distro for OpenTelemetry の詳細については、AWS Distro for OpenTelemetry を参照してください。AWS Distro for OpenTelemetry SDK の詳細については、「はじめに」を参照してください。

  このセクションには、次のフィールドを含めることができます。

  - grpc_endpoint - オプション。gRPC リモートプロシージャコールを使用して送信された OpenTelemetry トレースをリッスンするために使用する CloudWatch エージェントのアドレスを指定します。形式は ip:port です。このアドレスは、OpenTelemetry SDK の gRPC エクスポーター用に設定されたアドレスと一致する必要があります。

    このフィールドを省略した場合、127.0.0.1:4317 のデフォルトが使用されます。

  - http_endpoint - オプション。HTTP 経由で送信される OTLP トレースをリッスンするために CloudWatch エージェントが使用するアドレスを指定します。形式は ip:port です。このアドレスは、OpenTelemetry SDK の HTTP エクスポーター用に設定されたアドレスと一致する必要があります。

    このフィールドを省略した場合、127.0.0.1:4318 のデフォルトが使用されます。

- concurrency - オプション。トレースのアップロードに使用できる、X-Ray に対する同時呼び出しの最大数を指定します。デフォルト値は、「8」です。
- local_mode - オプション。true の場合、エージェントは Amazon EC2 インスタンスのメタデータを収集しません。デフォルトは です。false
- endpoint_override - オプション。 CloudWatch エージェントがトレースを送信するエンドポイントとして使用する FIPS エンドポイントまたはプライベートリンクを指定します。このフィー

ルドを指定してプライベートリンクを設定すると、Amazon VPC エンドポイントにトレースを送信できます。詳細については、「Amazon VPC とは」を参照してください。

endpoint_override の値は URL 文字列であることが必要です。

- region_override - オプション。X-Ray エンドポイント用に使用するリージョンを指定します。CloudWatch エージェントは、指定されたリージョンの X-Ray にトレースを送信します。このフィールドを省略すると、エージェントは Amazon EC2 インスタンスが配置されているリージョンにトレースを送信します。

  ここでリージョンを指定すると、設定ファイルの agent セクションの region パラメータの設定よりも優先されます。
- proxy_override - オプション。X-Ray にリクエストを送信する際に CloudWatch エージェントが使用するプロキシサーバーアドレスを指定します。プロキシサーバーのプロトコルは、このアドレスの一部として指定する必要があります。
- credentials – 異なる AWS アカウントにトレースを送信する際に使用する IAM ロールを指定します。指定した場合、このフィールドには 1 つのパラメータ role_arn が含まれています。
  - role_arn – 異なる AWS アカウントにトレースを送信する際の認証用 IAM ロールの ARN を指定します。詳細については、「別のアカウントへのメトリクス、ログ、トレースの送信」を参照してください。ここで指定した場合は、設定ファイルの role_arn セクションで指定された agent よりも優先されます (存在する場合)。

CloudWatch エージェント設定ファイル: 完全な例

Linux サーバーの完全な CloudWatch エージェント設定ファイルの例を次に示します。

収集するメトリクスについて measurement セクションにリストされた項目では、完全なメトリクス名、またはリソースのタイプに追加するメトリクス名の一部のみを指定できます。たとえば、reads セクションの diskio_reads セクションで measurement または diskio を指定すると、diskio_reads メトリクスが収集されます。

この例には、measurement セクションでメトリクスを指定するための両方の方法が含まれています。

```
{
  "agent": {
    "metrics_collection_interval": 10,
    "logfile": "/opt/aws/amazon-cloudwatch-agent/logs/amazon-cloudwatch-agent.log"
  },
```

```
      "metrics": {
        "namespace": "MyCustomNamespace",
        "metrics_collected": {
          "cpu": {
            "resources": [
              "*"
            ],
            "measurement": [
              {"name": "cpu_usage_idle", "rename": "CPU_USAGE_IDLE", "unit":
 "Percent"},
              {"name": "cpu_usage_nice", "unit": "Percent"},
              "cpu_usage_guest"
            ],
            "totalcpu": false,
            "metrics_collection_interval": 10,
            "append_dimensions": {
              "customized_dimension_key_1": "customized_dimension_value_1",
              "customized_dimension_key_2": "customized_dimension_value_2"
            }
          },
          "disk": {
            "resources": [
              "/",
              "/tmp"
            ],
            "measurement": [
              {"name": "free", "rename": "DISK_FREE", "unit": "Gigabytes"},
              "total",
              "used"
            ],
             "ignore_file_system_types": [
              "sysfs", "devtmpfs"
            ],
            "metrics_collection_interval": 60,
            "append_dimensions": {
              "customized_dimension_key_3": "customized_dimension_value_3",
              "customized_dimension_key_4": "customized_dimension_value_4"
            }
          },
          "diskio": {
            "resources": [
              "*"
            ],
            "measurement": [
```

```
            "reads",
            "writes",
            "read_time",
            "write_time",
            "io_time"
          ],
          "metrics_collection_interval": 60
        },
        "swap": {
          "measurement": [
            "swap_used",
            "swap_free",
            "swap_used_percent"
          ]
        },
        "mem": {
          "measurement": [
            "mem_used",
            "mem_cached",
            "mem_total"
          ],
          "metrics_collection_interval": 1
        },
        "net": {
          "resources": [
            "eth0"
          ],
          "measurement": [
            "bytes_sent",
            "bytes_recv",
            "drop_in",
            "drop_out"
          ]
        },
        "netstat": {
          "measurement": [
            "tcp_established",
            "tcp_syn_sent",
            "tcp_close"
          ],
          "metrics_collection_interval": 60
        },
        "processes": {
          "measurement": [
```

```
              "running",
              "sleeping",
              "dead"
            ]
          }
        },
        "append_dimensions": {
          "ImageId": "${aws:ImageId}",
          "InstanceId": "${aws:InstanceId}",
          "InstanceType": "${aws:InstanceType}",
          "AutoScalingGroupName": "${aws:AutoScalingGroupName}"
        },
        "aggregation_dimensions" : [["ImageId"], ["InstanceId", "InstanceType"],
 ["d1"],[]],
        "force_flush_interval" : 30
      },
      "logs": {
        "logs_collected": {
          "files": {
            "collect_list": [
              {
                "file_path": "/opt/aws/amazon-cloudwatch-agent/logs/amazon-cloudwatch-
agent.log",
                "log_group_name": "amazon-cloudwatch-agent.log",
                "log_stream_name": "amazon-cloudwatch-agent.log",
                "timezone": "UTC"
              },
              {
                "file_path": "/opt/aws/amazon-cloudwatch-agent/logs/test.log",
                "log_group_name": "test.log",
                "log_stream_name": "test.log",
                "timezone": "Local"
              }
            ]
          }
        },
        "log_stream_name": "my_log_stream_name",
        "force_flush_interval" : 15,
        "metrics_collected": {
           "kubernetes": {
                "enhanced_container_insights": true
      }
    }
  }
```

```
}
```

Windows Server を実行しているサーバーの完全な CloudWatch エージェント設定ファイルの例を次に示します。

```
{
      "agent": {
        "metrics_collection_interval": 60,
        "logfile": "c:\\ProgramData\\Amazon\\AmazonCloudWatchAgent\\Logs\\amazon-
cloudwatch-agent.log"
      },
      "metrics": {
        "namespace": "MyCustomNamespace",
        "metrics_collected": {
          "Processor": {
            "measurement": [
              {"name": "% Idle Time", "rename": "CPU_IDLE", "unit": "Percent"},
              "% Interrupt Time",
              "% User Time",
              "% Processor Time"
            ],
            "resources": [
              "*"
            ],
            "append_dimensions": {
              "customized_dimension_key_1": "customized_dimension_value_1",
              "customized_dimension_key_2": "customized_dimension_value_2"
            }
          },
          "LogicalDisk": {
            "measurement": [
              {"name": "% Idle Time", "unit": "Percent"},
              {"name": "% Disk Read Time", "rename": "DISK_READ"},
              "% Disk Write Time"
            ],
            "resources": [
              "*"
            ]
          },
          "customizedObjectName": {
            "metrics_collection_interval": 60,
            "customizedCounterName": [
              "metric1",
```

```
            "metric2"
          ],
          "resources": [
            "customizedInstances"
          ]
        },
        "Memory": {
          "metrics_collection_interval": 5,
          "measurement": [
            "Available Bytes",
            "Cache Faults/sec",
            "Page Faults/sec",
            "Pages/sec"
          ]
        },
        "Network Interface": {
          "metrics_collection_interval": 5,
          "measurement": [
            "Bytes Received/sec",
            "Bytes Sent/sec",
            "Packets Received/sec",
            "Packets Sent/sec"
          ],
          "resources": [
            "*"
          ],
          "append_dimensions": {
            "customized_dimension_key_3": "customized_dimension_value_3"
          }
        },
        "System": {
          "measurement": [
            "Context Switches/sec",
            "System Calls/sec",
            "Processor Queue Length"
          ]
        }
      },
      "append_dimensions": {
        "ImageId": "${aws:ImageId}",
        "InstanceId": "${aws:InstanceId}",
        "InstanceType": "${aws:InstanceType}",
        "AutoScalingGroupName": "${aws:AutoScalingGroupName}"
      },
```

```
        "aggregation_dimensions" : [["ImageId"], ["InstanceId", "InstanceType"],
 ["d1"],[]]
      },
      "logs": {
        "logs_collected": {
          "files": {
            "collect_list": [
              {
                "file_path": "c:\\ProgramData\\Amazon\\AmazonCloudWatchAgent\\Logs\
\amazon-cloudwatch-agent.log",
                "log_group_name": "amazon-cloudwatch-agent.log",
                "timezone": "UTC"
              },
              {
                "file_path": "c:\\ProgramData\\Amazon\\AmazonCloudWatchAgent\\Logs\
\test.log",
                "log_group_name": "test.log",
                "timezone": "Local"
              }
            ]
          },
          "windows_events": {
            "collect_list": [
              {
                "event_name": "System",
                "event_levels": [
                  "INFORMATION",
                  "ERROR"
                ],
                "log_group_name": "System",
                "log_stream_name": "System",
                "event_format": "xml"
              },
              {
                "event_name": "CustomizedName",
                "event_levels": [
                  "WARNING",
                  "ERROR"
                ],
                "log_group_name": "CustomizedLogGroup",
                "log_stream_name": "CustomizedLogStream",
                "event_format": "xml"
              }
            ]
```

```
          }
        },
        "log_stream_name": "example_log_stream_name"
      }
    }
```

CloudWatch エージェント設定ファイルを手動で保存するには

CloudWatch エージェント設定ファイルを手動で作成または編集する場合は、任意の名前を付けることができます。トラブルシューティングを簡単にするため、Linux サーバーでは、`/opt/aws/amazon-cloudwatch-agent/etc/amazon-cloudwatch-agent.json`、Windows Server を実行しているサーバーでは、`$Env:ProgramData\Amazon\AmazonCloudWatchAgent\amazon-cloudwatch-agent.json` という名前を付けることをお勧めします。ファイルを作成したら、エージェントを実行する他のサーバーにそのファイルをコピーできます。

CloudWatch エージェント設定ファイルの Systems Manager Parameter Store へのアップロード

SSM Agent を使用して CloudWatch エージェントをサーバーにインストールする場合は、CloudWatch エージェント設定ファイルを手動で編集した後、Systems Manager Parameter Store にアップロードできます。これを行うには、Systems Manager `put-parameter` コマンドを使用します。

Parameter Store にファイルを保存できるようにするには、十分なアクセス権限を持つ IAM ロールを使用する必要があります。詳細については、「CloudWatch エージェントで使用する IAM ロールとユーザーを作成する」を参照してください。

次のコマンドを使用します。ここで、*`parameter name`* は Parameter Store でこのファイルに使用する名前で、*`configuration_file_pathname`* は編集した設定ファイルのパスとファイル名です。

```
aws ssm put-parameter --name "parameter name" --type "String" --value
 file://configuration_file_pathname
```

CloudWatch Application Signals を有効にする

CloudWatch Application Signals を使用すると、AWS 上でアプリケーションを自動的に計測して、ビジネス目標に照らしてアプリケーションのパフォーマンスを追跡できます。Application Signals では、Java アプリケーションとその依存関係やエッジをアプリケーション中心の統合ビューで把握できます。詳細については、「Application Signals」を参照してください。

CloudWatch Application Signals は、CloudWatch エージェントを利用して、自動計測されたアプリケーションからメトリクスとトレースを受け取ります。必要に応じて、ルールを適用して高基数を減らしてから、処理済みのテレメトリを CloudWatch に公開できます。エージェント設定ファイルを使用すると、設定を特に Application Signals 向けにカスタマイズしてから CloudWatch エージェントに提供できます。エージェント設定ファイルでは `logs` セクション内の `metrics_collected` セクションに `app_signals` セクションがあり、CloudWatch エージェントが自動計測されたアプリケーションからメトリクスを受け取ることを指定しています。同様に、`traces` セクション内の `traces_collected` セクションに `app_signals` セクションがあり、CloudWatch エージェントが自動計測されたアプリケーションからトレースを受け取ることを指定しています。さらに、ここで説明しているように、必要に応じてカスタム設定ルールを渡して、公開する高基数テレメトリを減らすことができます。

- Amazon EKS クラスターの場合、Amazon CloudWatch Observability EKS アドオンをインストールすると、デフォルトで CloudWatch エージェントが自動計測されたアプリケーションからメトリクスとトレースの両方を受け取るようになります。カスタム設定ルールを渡すことにした場合は、(オプション) その他の設定 で説明しているように、別途必要な設定を行ってそのルールを作成または更新するときに、Amazon EKS アドオンにカスタムエージェント設定を渡します。
- Amazon EC2 などサポートされているその他のプラットフォームの場合、CloudWatch エージェントを起動するには、Application Signals を有効にするようにエージェントを設定しておく必要があります。そのためにはこの後で説明しているように、`app_signals` セクションを指定し、さらに必要に応じてカスタム設定ルールを指定します。

次に、CloudWatch エージェント設定ファイルにあるフィールドのうち、CloudWatch Application Signals に関連するフィールドの概要を示します。

- `logs`
  - `metrics_collected` - このフィールドには、エージェントにログ収集を指示するセクションを含めることができます。CloudWatch Application Signals と Container Insights のオブザーバビリティを Amazon EKS 向けに強化するといったユースケースを利用できるようになります。

> **Note**
>
> 以前このセクションは、エージェントが埋め込みメトリクス形式のログを収集するように指定するためにも使用されていました。それらの設定はもう必要ありません。

- app_signals (オプション) CloudWatch Application Signals を有効にして、自動計測されたアプリケーションからメトリクスを受け取るよう指定し、CloudWatch Application Signals を利用しやすくします。
  - rules (オプション) 高基数を使用する状況に対応できるように、条件に従ってメトリクスとトレースを選択してアクションを適用する一連のルール。各ルールに次のフィールドの要素を含めることができます。
    - rule_name (オプション) ルールの名前。
    - selectors (オプション) メトリクスとトレースの一連のディメンションマッチャー。各セレクターでは、以下のフィールドを指定する必要があります。
      - dimension selectors が空でない場合は必須です。フィルターとして使用するメトリクスとトレースのディメンションを指定します。
      - match selectors が空でない場合は必須です。指定したディメンションの値のマッチングに使用されるワイルドカードパターン。
    - action (オプション) 指定したセレクターに一致したメトリクスとトレースに適用するアクション。action の値は、次のキーワードのいずれかにする必要があります。
      - keep メトリクスとトレースを selectors に一致した場合にのみ CloudWatch に送信するように指定します。
      - drop selectors に一致するメトリクスとトレースをドロップするように指定します。
      - replace selectors に一致するメトリクスとトレースのディメンションを置き換えるように指定します。replacements セクションに従って置き換えられます。
    - replacements action が replace の場合は必須です。action が replace の場合に、指定した selectors に一致するメトリクスとトレースに適用される一連のディメンションと値のペア。置き換えを行うには、以下のフィールドを指定する必要があります。
      - target_dimension replacements が空でない場合は必須です。置き換える必要があるディメンションを指定します。
      - value replacements が空でない場合は必須です。置き換え対象の値 target_dimension をこの値に置き換えます。
  - limiter (オプション) このセクションを使用して、Application Signals が CloudWatch に送信するメトリクスとディメンションの数を制限し、コストを最適化します。
    - disabled (オプション) true の場合、メトリクス制限機能は無効になります。デフォルトは です。false

- drop_threshold (オプション) 1 つの CloudWatch エージェントがエクスポートできる、1 つのローテーション間隔におけるサービスごとの個別メトリクスの最大数。デフォルトは 500 です。
- rotation_interval (オプション) リミッターが区別してカウントするためにメトリクスレコードをリセットする間隔。これは、一連の数字と単位サフィックスを使用する文字列として表されます。分数がサポートされています。サポートされている単位サフィックスは、s、m、h、ms、us、および ns です。

  デフォルトは 1 時間の 1h です。
- log_dropped_metrics (オプション) Application Signals のメトリクスがドロップされたときに、エージェントが CloudWatch エージェントログにログを書き込む必要があるかどうかを指定します。デフォルト: false。

> **Note**
>
> このログ記録を有効にするには、agent セクションの debug パラメータも true に設定する必要があります。

- traces
  - traces_collected
    - app_signals オプション。このフィールドを指定すると、CloudWatch Application Signals を利用しやすくするために、CloudWatch エージェントが自動計測されたアプリケーションからメトリクスを受け取るようにできます。

> **Note**
>
> カスタムの app_signals ルールは、logs セクションに含まれている metrics_collected セクションの下に指定している場合でも、traces_collected セクションに暗黙的に適用されます。同じルールセットがメトリクスとトレースの両方に適用されます。

アクションの異なるルールが複数ある場合は、keep、drop、replace という順序で適用されます。

次に、CloudWatch エージェント設定ファイルでカスタムルールを適用する場合の設定例を示します。

```
{
  "logs": {
    "metrics_collected": {
      "app_signals": {
        "rules": [
          {
            "rule_name": "keep01",
            "selectors": [
              {
                "dimension": "Service",
                "match": "pet-clinic-frontend"
              },
              {
                "dimension": "RemoteService",
                "match": "customers-service"
              }
            ],
            "action": "keep"
          },
          {
            "rule_name": "drop01",
            "selectors": [
              {
                "dimension": "Operation",
                "match": "GET /api/customer/owners/*"
              }
            ],
            "action": "drop"
          },
          {
            "rule_name": "replace01",
            "selectors": [
              {
                "dimension": "Operation",
                "match": "PUT /api/customer/owners/*/pets/*"
              },
              {
                "dimension": "RemoteOperation",
                "match": "PUT /owners"
              }
```

```
            ],
            "replacements": [
              {
                "target_dimension": "Operation",
                "value": "PUT /api/customer/owners/{ownerId}/pets{petId}"
              }
            ],
            "action": "replace"
          }
        ]
      }
    }
  },
  "traces": {
    "traces_collected": {
      "app_signals": {}
    }
  }
}
```

この例の設定ファイルの場合、rules は以下のように処理されます。

1. ルール keep01 により、ディメンション Service が pet-clinic-frontend で、ディメンション RemoteService が customers-service であるメトリクスとトレースが保持されます。
2. keep01 の適用後、処理されたメトリクスとトレースを対象にルール drop01 が適用され、ディメンション Operation が GET /api/customer/owners/* であるメトリクスとトレースがドロップされます。
3. drop01 の適用後、処理されたメトリクスとトレースを対象にルール replace01 が適用され、ディメンション Operation が PUT /api/customer/owners/*/pets/* で、ディメンション RemoteOperation が PUT /owners であるメトリクスとトレースが更新されます。その結果、Operation ディメンションが PUT /api/customer/owners/{ownerId}/pets{petId} に変更されます。

メトリクスの制限を 100 に変更し、ドロップされたメトリクスのログ記録を有効にして、ローテーション間隔を 2 時間に設定することで、Application Signals の基数を管理する CloudWatch 設定ファイルの完全な例を以下に示します。

```
{
    "logs": {
```

```
        "metrics_collected": {
            "app_signals": {
                "limiter": {
                    "disabled": false,
                    "drop_threshold": 100,
                    "rotation_interval": "2h",
                    "log_dropped_metrics": true
                }
            }
        },
        "traces": {
            "traces_collected": {
                "app_signals": {}
            }
        }
    }
}
```

ネットワークパフォーマンスメトリクスの収集

Elastic Network Adapter (ENA) を使用する Linux で実行されている EC2 インスタンスは、ネットワークパフォーマンスメトリクスを発行します。バージョン 1.246396.0 以降の CloudWatch エージェントでは、これらのネットワークパフォーマンスメトリクスを CloudWatch にインポートできます。これらのネットワークパフォーマンスメトリクスを CloudWatch にインポートすると、CloudWatch カスタムメトリクスとして課金されます。

ENA ドライバーの詳細については、「Linux インスタンスにおける Elastic Network Adapter (ENA) を使用した拡張ネットワーキングの有効化」および「Windows インスタンスにおける Elastic Network Adapter (ENA) を使用した拡張ネットワーキングの有効化」をご参照ください。

ネットワークパフォーマンスメトリクスの収集の設定方法は、Linux サーバーと Windows サーバーで異なります。

以下の表では、ENA アダプターによって有効化されたこれらのネットワークパフォーマンスメトリクスを示しています。CloudWatch エージェントがこれらのメトリクスを Linux インスタンスから CloudWatch にインポートすると、これらの各メトリクス名の先頭に `ethtool_` が付加されます。

メトリクス	説明
Linux サーバー上の名前: **bw_in_allowance_exceeded** Windows サーバー上の名前: **Aggregate inbound BW allowance exceeded**	インバウンド集計の帯域幅がインスタンスの最大値を超えたために、キューおよび (または) ドロップされたパケットの数。 このメトリクスは、CloudWatch エージェント設定ファイルの `ethtool` セクションの `metrics_collected` サブセクションにリストされている場合にのみ収集されます。詳細については、ネットワークパフォーマンスメトリクスの収集 を参照してください。 単位: なし
Linux サーバー上の名前: **bw_out_allowance_exceeded** Windows サーバー上の名前: **Aggregate outbound BW allowance exceeded**	アウトバウンド集計の帯域幅がインスタンスの最大値を超えたために、キューおよび (または) ドロップされたパケットの数。 このメトリクスは、CloudWatch エージェント設定ファイルの `ethtool` セクションの `metrics_collected` サブセクションにリストされている場合にのみ収集されます。詳細については、ネットワークパフォーマンスメトリクスの収集 を参照してください。 単位: なし
Linux サーバー上の名前: **conntrack_allowance_available** Windows サーバー上の名前: **Available connection tracking allowance**	そのインスタンスタイプの Connections Tracked 許容量に達する前にインスタンスが確立できる接続トラッキング数をレポートします。このメトリクスは、バージョン 2.8.1 以降の Elastic Network Adapter (ENA) 用の Linux ドライバーを使用する Nitro ベースの EC2 インスタンス、およびバージョン 2.6.0 以降の Elastic Network Adapter (ENA) 用の Windows ドライバーを使用するコンピュータでのみ使用できます。

メトリクス	説明
	このメトリクスは、CloudWatch エージェント設定ファイルの `ethtool` セクションの `metrics_collected` サブセクションにリストされている場合にのみ収集されます。詳細については、ネットワークパフォーマンスメトリクスの収集 を参照してください。 単位: なし
Linux サーバー上の名前: **`ena_srd_mode`** Windows サーバー上の名前: **`ena srd mode`**	ENA Express のどの機能が有効になっているかを説明します。ENA Express の詳細については、「Linux インスタンスで ENA Express を使用してネットワークパフォーマンスを向上させる」を参照してください。値は次のとおりです。 • `0` = ENA Express がオフ、UDP がオフ • `1` = ENA Express がオン、UDP がオフ • `2` = ENA Express がオフ、UDP がオン **Note** これは、ENA Express が最初に有効になっており、UDP がそれを使用するように設定されている場合にのみ発生します。UDP トラフィックの以前の値は保持されます。 • `3` = ENA Express がオン、UDP がオン

メトリクス	説明
Linux サーバー上の名前: **ena_srd_eligible_tx_pkts** Windows サーバー上の名前: **ena srd eligible tx pkts**	一定期間内に送信された AWS Scalable Reliable Datagram (SRD) の資格要件を満たすネットワークパケットの数。次のようになります。 • 送信側と受信側の両方のインスタンスタイプがサポートされています。 • 送信側と受信側の両方のインスタンスに ENA Express が設定されている必要があります。 • 送信側と受信側のインスタンスは同じサブネットに存在する必要があります。 • インスタンス間のネットワークパスには、ミドルウェアボックスを含めないようにしてください。ENA Express は現在、ミドルウェアボックスをサポートしていません。
Linux サーバー上の名前: **ena_srd_tx_pkts** Windows サーバー上の名前: **ena srd tx pkts**	一定期間内に送信した SRD パケット数。
Linux サーバー上の名前: **ena_srd_rx_pkts** Windows サーバー上の名前: **ena srd rx pkts**	一定期間内に受信した SRD パケット数。
Linux サーバー上の名前: **ena_srd_resource_utilization** Windows サーバー上の名前: **ena srd resource utilization**	インスタンスが消費した同時 SRD 接続の最大許容メモリ使用量の割合。

メトリクス	説明
Linux サーバー上の名前: **linklocal_allowance_exceeded** Windows サーバー上の名前: **Link local packet rate allowance exceeded**	ローカルプロキシサービスへのトラフィックの PPS がネットワークインターフェイスの最大値を超えたためにドロップされたパケットの数。これは、DNS サービス、インスタンスメタデータサービス、および Amazon Time Sync Service へのトラフィックに影響します。 このメトリクスは、CloudWatch エージェント設定ファイルの `ethtool` セクションの `metrics_collected` サブセクションにリストされている場合にのみ収集されます。詳細については、ネットワークパフォーマンスメトリクスの収集 を参照してください。 単位: なし
Linux サーバー上の名前: **linklocal_allowance_exceeded** Windows サーバー上の名前: **Link local packet rate allowance exceeded**	ローカルプロキシサービスへのトラフィックの PPS がネットワークインターフェイスの最大値を超えたためにドロップされたパケットの数。これは、DNS サービス、インスタンスメタデータサービス、および Amazon Time Sync Service へのトラフィックに影響します。 このメトリクスは、CloudWatch エージェント設定ファイルの `ethtool` セクションの `metrics_collected` サブセクションにリストされている場合にのみ収集されます。詳細については、ネットワークパフォーマンスメトリクスの収集 を参照してください。 単位: なし

メトリクス	説明
Linux サーバー上の名前: **pps_allowance_exceeded** Windows サーバー上の名前: **PPS allowance exceeded**	双方向 PPS がインスタンスの最大値を超えたために、キューおよび (または) ドロップされたパケットの数。 このメトリクスは、CloudWatch エージェント設定ファイルの ethtool セクションの metrics_collected サブセクションにリストされている場合にのみ収集されます。詳細については、ネットワークパフォーマンスメトリクスの収集 を参照してください。 単位: なし

Linux の設定

Linux サーバーでは、ethtool プラグインを使用すると、ネットワークパフォーマンスメトリクスを CloudWatch にインポートできます。

ethtool は、Linux サーバー上のイーサネットデバイスに関する統計を収集できる標準の Linux ユーティリティです。収集される統計情報は、ネットワークデバイスとドライバーによって異なります。これらの統計情報の例には、tx_cnt、rx_bytes、tx_errors、align_errors などがあります。CloudWatch エージェントで ethtool プラグインを使用する場合、このセクションで前述した EC2 ネットワークパフォーマンスメトリクスとともに、これらの統計を CloudWatch にインポートすることもできます。

> **Tip**
>
> オペレーティングシステムとネットワークデバイスで利用可能な統計を確認するには、ethtool -S コマンドを使用します。

CloudWatch エージェントがメトリクスを CloudWatch にインポートすると、インポートされたすべてのメトリクスの名前に ethtool_ プレフィックスが追加されます。したがって、標準の ethtool 統計 rx_bytes は CloudWatch で ethtool_rx_bytes として呼び出され、EC2 ネットワークパフォーマンスメトリクス bw_in_allowance_exceeded は CloudWatch で ethtool_bw_in_allowance_exceeded として呼び出されます。

Linux サーバーで ethtool メトリクスをインポートするには、CloudWatch エージェント設定ファイルの ethtool セクションに metrics_collected セクションを追加します。ethtool セクションでは、次のサブセクションを含めることができます。

- interface_include – このセクションを含めると、エージェントは、このセクションにリストされている名前を持つインターフェイスからのみメトリクスを収集します。このセクションを省略すると、interface_exclude にリストされていないすべてのイーサネットインターフェイスからメトリクスが収集されます。

  デフォルトのイーサネットインターフェイスは eth0 です。
- interface_exclude – このセクションを含める場合は、メトリクスを収集しないイーサネットインターフェイスをリストします。

  ethtool プラグインは、常にループバックインターフェイスを無視します。
- metrics_include – このセクションには、CloudWatch にインポートするメトリクスがリストされます。ethtool によって収集された標準統計情報と、Amazon EC2 高解像度ネットワークメトリクスの両方を含めることができます。

以下の例では、CloudWatch エージェント設定ファイルの一部が表示されています。この設定では、インターフェイスから標準の ethtool メトリクス rx_packets および tx_packets、ならびに eth1 インターフェイスのみから Amazon EC2 ネットワークパフォーマンスメトリクスが収集されます。

CloudWatch エージェント設定ファイルの詳細については、「CloudWatch エージェント設定ファイルを手動で作成または編集する」を参照してください。

```
"metrics": {
    "append_dimensions": {
      "InstanceId": "${aws:InstanceId}"
    },
    "metrics_collected": {
      "ethtool": {
        "interface_include": [
          "eth1"
        ],
        "metrics_include": [
          "rx_packets",
          "tx_packets",
          "bw_in_allowance_exceeded",
          "bw_out_allowance_exceeded",
```

```
            "conntrack_allowance_exceeded",
            "linklocal_allowance_exceeded",
            "pps_allowance_exceeded"
          ]
       }
    }
}
```

Windows セットアップ

Windows サーバーでは、ネットワークパフォーマンスメトリクスは、CloudWatch エージェントが既にメトリクスを収集している Windows パフォーマンスカウンターから入手できます。そのため、Windows サーバーからこれらのメトリックを収集するのにプラグインは必要ありません。

次は、Windows からネットワークパフォーマンスメトリクスを収集するためのサンプル設定ファイルです。CloudWatch エージェント設定ファイルの編集の詳細については、「CloudWatch エージェント設定ファイルを手動で作成または編集する」を参照してください。

```
{
    "metrics": {
        "append_dimensions": {
            "InstanceId": "${aws:InstanceId}"
        },
        "metrics_collected": {
            "ENA Packets Shaping": {
                "measurement": [
                    "Aggregate inbound BW allowance exceeded",
                    "Aggregate outbound BW allowance exceeded",
                    "Connection tracking allowance exceeded",
                    "Link local packet rate allowance exceeded",
                    "PPS allowance exceeded"
                ],
                "metrics_collection_interval": 60,
                "resources": [
                    "*"
                ]
            }
        }
    }
}
```

ネットワークパフォーマンスメトリクスの表示

ネットワークパフォーマンスメトリクスを CloudWatch にインポートした後、これらのメトリクスを時系列グラフとして表示し、これらのメトリクスをモニターリングし、指定したしきい値に違反した場合に通知できるアラームを作成できます。以下の手順では、ethtool メトリクスを時系列グラフとして表示する方法を示します。アラームの設定の詳細については、「Amazon CloudWatch でのアラームの使用」を参照してください。

これらのメトリクスはすべて集計カウンターであるため、`RATE(METRICS())` などの CloudWatch メトリクス数学関数を使用して、これらのメトリクスのレートをグラフで計算したり、アラームを設定したりできます。メトリクス数学関数の詳細については、「Metric Math を使用する」を参照してください。

ネットワークパフォーマンスメトリクスを CloudWatch コンソールに表示するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで [Metrics (メトリクス)] を選択します。
3. エージェントによって収集されたメトリクスの名前空間を選択します。デフォルトでは、これは CWAgent ですが、CloudWatch エージェント設定ファイルで別の名前空間を指定している場合があります。
4. メトリクスのディメンション (例: インスタンス別メトリクス) を選択します。
5. [All metrics] タブには、名前空間内のそのディメンションのメトリクスがすべて表示されます。以下の操作を行うことができます。

    a. メトリクスをグラフ表示するには、メトリクスの横にあるチェックボックスを選択します。すべてのメトリクスを選択するには、テーブルの見出し行にあるチェックボックスを選択します。

    b. テーブルを並べ替えるには、列見出しを使用します。

    c. リソースでフィルタするには、リソース ID を選択し、[Add to search] (検索に追加) を選択します。

    d. メトリクスでフィルターするには、メトリクス名を選択し、[Add to search] (検索に追加) を選択します。

6. (オプション) このグラフを CloudWatch ダッシュボードに追加するには、[Actions] (アクション)、[Add to dashboard] (ダッシュボードに追加) の順に選択します。

NVIDIA GPU メトリクスを収集する

CloudWatch エージェントを使用して、Linux サーバーから NVIDIA GPU メトリクスを収集できます。これを設定するには、CloudWatch エージェント設定ファイルの `metrics_collected` セクション内に `nvidia_gpu` セクションを追加します。詳細については、「Linux のセクション」を参照してください。

さらに、インスタンスには NVIDIA ドライバーがインストールされている必要があります。NVIDIA ドライバーは、一部の Amazon マシンイメージ (AMI) にプリインストールされています。それ以外の場合は、ドライバーを手動でインストールできます。詳細については、「Linux インスタンスへの NVIDIA ドライバーのインストール」を参照してください。

次のメトリクスを収集できます。これらのメトリクススはすべて CloudWatch `Unit` なしで収集されますが、CloudWatch エージェント設定ファイルにパラメータを追加することで、各メトリクスの単位を指定できます。詳細については、「Linux のセクション」を参照してください。

メトリクス	CloudWatch のメトリクス名	説明
`utilization_gpu`	`nvidia_smi_utilization_gpu`	GPU 上の 1 つ以上のカーネルが実行されていた過去のサンプル期間における時間の割合。
`temperature_gpu`	`nvidia_smi_temperature_gpu`	コア GPU 温度 (摂氏)。
`power_draw`	`nvidia_smi_power_draw`	ボード全体で最後に測定された電力消費量 (ワット)。
`utilization_memory`	`nvidia_smi_utilization_memory`	グローバル (デバイス) メモリの読み取りまたは書き込みが行われていた過去のサンプル期間における時間の割合。
`fan_speed`	`nvidia_smi_fan_speed`	デバイスのファンが現在意図している最大ファン速度に対する割合。
`memory_total`	`nvidia_smi_memory_total`	レポートされた合計メモリ (MB)。

メトリクス	CloudWatch のメトリクス名	説明
memory_used	nvidia_smi_memory_used	使用済みメモリ (MB)。
memory_free	nvidia_smi_memory_free	利用可能なメモリ (MB)。
pcie_link_gen_current	nvidia_smi_pcie_link_gen_current	現在のリンク生成。
pcie_link_width_current	nvidia_smi_pcie_link_width_current	現在のリンク幅。
encoder_stats_session_count	nvidia_smi_encoder_stats_session_count	現在のエンコーダセッション数。
encoder_stats_average_fps	nvidia_smi_encoder_stats_average_fps	エンコードフレーム/秒の移動平均。
encoder_stats_average_latency	nvidia_smi_encoder_stats_average_latency	エンコードレイテンシーの移動平均 (マイクロ秒)。
clocks_current_graphics	nvidia_smi_clocks_current_graphics	グラフィックス (シェーダー) クロックの現在の周波数。
clocks_current_sm	nvidia_smi_clocks_current_sm	Streaming Multiprocessor (SM) クロックの現在の周波数。

メトリクス	CloudWatch のメトリクス名	説明
clocks_current_memory	nvidia_smi_clocks_current_memory	メモリクロックの現在の周波数。
clocks_current_video	nvidia_smi_clocks_current_video	動画 (エンコーダとデコーダ) クロックの現在の周波数。

これらのメトリクスはすべて、次のディメンションで収集されます。

ディメンション	説明
index	このサーバー上の GPU の一意の識別子。デバイスの NVIDIA Management Library (NVML) インデックスを表します。
name	GPU の種類。例えば、NVIDIA Tesla A100
host	サーバーホスト名。

procstat プラグインでプロセスメトリクスを収集する

procstat プラグインでは、個別のプロセスからメトリクスを収集できます。このプラグインは、Linux サーバーと、サポート対象バージョンの Windows Server を実行するサーバーでサポートされます。

トピック

procstat 向けの CloudWatch エージェントの設定

procstat プラグインを使用するには、CloudWatch エージェント設定ファイルの `procstat` セクションに `metrics_collected` セクションを追加します。モニターリングするプロセスを指定するには 3 つの方法があります。これらの方法のうち使用できるのは 1 つのみですが、この 1 つの方法を使用して複数のプロセスを指定してモニターリングできます。

- `pid_file`: 作成するプロセス識別番号 (PID) ファイルの名前でプロセスを選択します。
- `exe`: 正規表現の照合ルールを使用して、指定した文字列と一致するプロセス名のプロセスを選択します。一致は「含む」一致です。つまり、一致する用語として `agent` を指定した場合、`cloudwatchagent` のような名前を持つプロセスは、その用語に一致します。詳細については、「Syntax」を参照してください。
- `pattern`: プロセスの起動に使用するコマンドラインでプロセスを選択します。正規表現の照合ルールを使用して指定した文字列と一致するコマンドラインを持つすべてのプロセスが選択されます。コマンドで使用されるパラメータやオプションも含めて、コマンドライン全体がチェックされます。

  一致は「含む」一致です。つまり、一致する用語として `-c` を指定した場合、`-config` のようなパラメータを持つプロセスは、その用語に一致します。
- `drop_original_metrics` - オプション。`metrics` セクションの `aggregation_dimensions` フィールドを使用してメトリクスを集計結果にロールアップしている場合、デフォルトでは、エージェントは集計メトリクスと、ディメンションの値ごとに分離された元のメトリクスの両方を送信します。元のメトリクスを CloudWatch に送信したくない場合は、メトリクスのリストを使用してこのパラメータを指定できます。このパラメータとともに指定されたメトリクスには、CloudWatch にレポートされるディメンションごとのメトリクスがありま

せん。代わりに、集計されたメトリクスのみがレポートされます。これにより、エージェントが収集するメトリクスの数が減り、コストが削減されます。

以上の複数のセクションを含めた場合でも、CloudWatch エージェントで使用される方法は 1 つのみです。複数のセクションを指定すると、CloudWatch エージェントは pid_file セクションを使用します (ある場合)。これが存在しない場合は、exe セクションを使用します。

Linux サーバーの場合、exe セクションまたは pattern セクションで指定した文字列は正規表現として評価されます。Windows Server を実行するサーバーの場合、これらの文字列は WMI クエリとして評価されます。文字列の例は pattern: "%apache%" などです。詳細については、「LIKE Operator」を参照してください。

どの方法を使用する場合でも、省略可能な metrics_collection_interval パラメータを含めることができます。このパラメータは、これらのメトリクスを収集する間隔を秒単位で指定します。このパラメータを省略すると、デフォルト値の 60 秒が使用されます。

以下のセクションの例では、procstat セクションのみをエージェント設定ファイルの metrics_collected セクションに追加しています。実際の設定ファイルでは、他のセクションも metrics_collected に追加できます。詳細については、「 CloudWatch エージェント設定ファイルを手動で作成または編集する」を参照してください。

pid_file による設定

次の例の procstat セクションでは、PID ファイルの example1.pid および example2.pid の作成プロセスをモニターリングします。プロセスごとに異なるメトリクスが収集されます。example2.pid の作成プロセスからのメトリクスは、10 秒間隔で収集されます。example1.pid プロセスからのメトリクスは、デフォルト値の 60 秒間隔で収集されます。

```
{
    "metrics": {
        "metrics_collected": {
            "procstat": [
                {
                    "pid_file": "/var/run/example1.pid",
                    "measurement": [
                        "cpu_usage",
                        "memory_rss"
                    ]
                },
                {
```

```
                    "pid_file": "/var/run/example2.pid",
                    "measurement": [
                        "read_bytes",
                        "read_count",
                        "write_bytes"
                    ],
                    "metrics_collection_interval": 10
                }
            ]
        }
    }
}
```

exe による設定

次の例の procstat セクションでは、文字列の agent または plugin と一致する名前を持つすべてのプロセスをモニターリングします。各プロセスから同じメトリクスが収集されます。

```
{
    "metrics": {
        "metrics_collected": {
            "procstat": [
                {
                    "exe": "agent",
                    "measurement": [
                        "cpu_time",
                        "cpu_time_system",
                        "cpu_time_user"
                    ]
                },
                {
                    "exe": "plugin",
                    "measurement": [
                        "cpu_time",
                        "cpu_time_system",
                        "cpu_time_user"
                    ]
                }
            ]
        }
    }
}
```

pattern による設定

次の例の procstat セクションでは、文字列の config または -c と一致するコマンドラインを持つすべてのプロセスをモニターリングします。各プロセスから同じメトリクスが収集されます。

```
{
    "metrics": {
        "metrics_collected": {
            "procstat": [
                {
                    "pattern": "config",
                    "measurement": [
                        "rlimit_memory_data_hard",
                        "rlimit_memory_data_soft",
                        "rlimit_memory_stack_hard",
                        "rlimit_memory_stack_soft"
                    ]
                },
                {
                    "pattern": "-c",
                    "measurement": [
                        "rlimit_memory_data_hard",
                        "rlimit_memory_data_soft",
                        "rlimit_memory_stack_hard",
                        "rlimit_memory_stack_soft"
                    ]
                }
            ]
        }
    }
}
```

Procstat で収集されるメトリクス

次の表は、procstat プラグインで収集できるメトリクスの一覧です。

CloudWatch エージェントは、以下のメトリクス名の先頭に procstat を追加します。収集元が Linux サーバーであるか、Windows Server を実行するサーバーであるかに応じて構文は異なります。たとえば、cpu_time メトリクスは、収集元が Linux である場合は procstat_cpu_time と表示され、収集元が Windows Server である場合は procstat cpu_time と表示されます。

メトリクス名	次で利用可能	説明
cpu_time	Linux	プロセスで CPU を使用する時間。このメトリクスは、1/100 秒単位で測定されます。 単位: 個
cpu_time_guest	Linux	プロセスがゲストモードになっている時間です。このメトリクスは、1/100 秒単位で測定されます。 タイプ: 浮動小数点 単位: なし
cpu_time_guest_nice	Linux	プロセスが nice ゲストで実行されている時間です。このメトリクスは、1/100 秒単位で測定されます。 タイプ: 浮動小数点

メトリクス名	次で利用可能	説明
		単位: なし
cpu_time_idle	Linux	プロセスがアイドルモードになっている時間です。このメトリクスは、1/100 秒単位で測定されます。 タイプ: 浮動小数点 単位: なし
cpu_time_iowait	Linux	プロセスが I/O 操作の完了を待機している時間です。このメトリクスは、1/100 秒単位で測定されます。 タイプ: 浮動小数点 単位: なし

メトリクス名	次で利用可能	説明
cpu_time_irq	Linux	プロセスが中断を処理している時間です。このメトリクスは、1/100 秒単位で測定されます。 タイプ: 浮動小数点 単位: なし
cpu_time_nice	Linux	プロセスが nice モードになっている時間です。このメトリクスは、1/100 秒単位で測定されます。 タイプ: 浮動小数点 単位: なし

メトリクス名	次で利用可能	説明
cpu_time_soft_irq	Linux	プロセスがソフトウェアの中断を処理している時間です。このメトリクスは、1/100 秒単位で測定されます。 タイプ: 浮動小数点 単位: なし
cpu_time_steal	Linux	仮想化環境で実行中に、他のオペレーティングシステムでの実行に費やされる時間です。このメトリクスは、1/100 秒単位で測定されます。 タイプ: 浮動小数点 単位: なし

メトリクス名	次で利用可能	説明
cpu_time_stolen	Linux、Windows Server	プロセスが盗まれた時間になっている時間です。これは、仮想化環境で他のオペレーティングシステムに費やされる時間です。このメトリクスは、1/100 秒単位で測定されます。 タイプ: 浮動小数点 単位: なし
cpu_time_system	Linux、Windows Server、macOS	プロセスがシステムモードになっている時間。このメトリクスは、1/100 秒単位で測定されます。 タイプ: 浮動小数点 単位: カウント

メトリクス名	次で利用可能	説明
cpu_time_user	Linux、Windows Server、macOS	プロセスがユーザーモードになっている時間。このメトリクスは、1/100 秒単位で測定されます。 単位: カウント
cpu_usage	Linux、Windows Server、macOS	任意の容量でプロセスがアクティブになっている時間の割合。 単位: パーセント
memory_data	Linux、macOS	プロセスで使用するデータ用メモリの量。 単位: バイト
memory_locked	Linux、macOS	プロセスでロックされているメモリの量。 単位: バイト

メトリクス名	次で利用可能	説明
memory_rss	Linux、Windows Server、macOS	プロセスが使用している実際のメモリ (常駐セット) の量。 単位: バイト
memory_stack	Linux、macOS	プロセスで使用されているスタックメモリの量。 単位: バイト
memory_swap	Linux、macOS	プロセスで使用されているスワップメモリの量。 単位: バイト
memory_vms	Linux、Windows Server、macOS	プロセスで使用されている仮想メモリの量。 単位: バイト
num_fds	Linux	このプロセスが開いているファイルディスクリプタの数です。 単位: なし

メトリクス名	次で利用可能	説明
num_threads	Linux、Windows、macOS	このプロセス内におけるスレッドの数です。 単位: なし
pid	Linux、Windows Server、macOS	プロセス識別子 (ID)。 単位: なし
pid_count	Linux、Windows Server、macOS	プロセスに関連付けられたプロセス ID の数。 このメトリクスのフルネームは、Linux サーバーおよび macOS コンピュータでは procstat_lookup_pid_count 、Windows サーバーでは procstat_lookup pid_count です。 単位: なし

メトリクス名	次で利用可能	説明
read_bytes	Linux、Windows Server	プロセスがディスクから読み取ったバイト数。 単位: バイト
write_bytes	Linux、Windows Server	プロセスがディスクに書き込んだバイト数。 単位: バイト
read_count	Linux、Windows Server	プロセスが実行したディスク読み込みオペレーションの数。 単位: なし
rlimit_realtime_priority_hard	Linux	このプロセスに設定できる最大リアルタイム優先度のハード制限です。 単位: なし

メトリクス名	次で利用可能	説明
rlimit_realtime_priority_soft	Linux	このプロセスに設定できる最大リアルタイム優先度のソフト制限です。 単位: なし
rlimit_signals_pending_hard	Linux	このプロセスでキューに入れられる最大シグナル数のハード制限です。 単位: なし
rlimit_signals_pending_soft	Linux	このプロセスでキューに入れられる最大シグナル数のソフト制限です。 単位: なし
rlimit_nice_priority_hard	Linux	このプロセスで設定できる最大 nice 優先度のハード制限です。 単位: なし

メトリクス名	次で利用可能	説明
rlimit_nice_priority_soft	Linux	このプロセスで設定できる最大 nice 優先度のソフト制限です。 単位: なし
rlimit_num_fds_hard	Linux	このプロセスが開くことができるファイルディスクリプタの最大数のハード制限です。 単位: なし
rlimit_num_fds_soft	Linux	このプロセスが開くことができるファイルディスクリプタの最大数のソフト制限です。 単位: なし
write_count	Linux、Windows Server	プロセスが実行したディスク書き込みオペレーションの数。 単位: なし

メトリクス名	次で利用可能	説明
involuntary_context_switches	Linux	プロセスのコンテキストが意図せずに切り替えられた回数。 単位: なし
voluntary_context_switches	Linux	プロセスのコンテキストが意図的に切り替えられた回数。 単位: なし
realtime_priority	Linux	プロセスで現在使用されているリアルタイム優先度。 単位: なし
nice_priority	Linux	プロセスで現在使用されている nice 優先度。 単位: なし
signals_pending	Linux	プロセスによる処理待ち中のシグナルの数。 単位: なし

メトリクス名	次で利用可能	説明
rlimit_cpu_time_hard	Linux	プロセスの CPU 時間のハードリソース制限。 単位: なし
rlimit_cpu_time_soft	Linux	プロセスの CPU 時間のソフトリソース制限。 単位: なし
rlimit_file_locks_hard	Linux	プロセスのファイルロックのハードリソース制限。 単位: なし
rlimit_file_locks_soft	Linux	プロセスのファイルロックのソフトリソース制限。 単位: なし
rlimit_memory_data_hard	Linux	プロセスのデータ用メモリに関するハードリソース制限。 単位: バイト

メトリクス名	次で利用可能	説明
rlimit_memory_data_soft	Linux	プロセスのデータ用メモリに関するソフトリソース制限。 単位: バイト
rlimit_memory_locked_hard	Linux	プロセスのロックされたメモリに関するハードリソース制限。 単位: バイト
rlimit_memory_locked_soft	Linux	プロセスのロックされたメモリに関するソフトリソース制限。 単位: バイト
rlimit_memory_rss_hard	Linux	プロセスの物理メモリに関するハードリソース制限。 単位: バイト
rlimit_memory_rss_soft	Linux	プロセスの物理メモリに関するソフトリソース制限。 単位: バイト

メトリクス名	次で利用可能	説明
rlimit_memory_stack_hard	Linux	プロセススタックに関するハードリソース制限。 単位: バイト
rlimit_memory_stack_soft	Linux	プロセススタックに関するソフトリソース制限。 単位: バイト
rlimit_memory_vms_hard	Linux	プロセスの仮想メモリに関するハードリソース制限。 単位: バイト
rlimit_memory_vms_soft	Linux	プロセスの仮想メモリに関するソフトリソース制限。 単位: バイト

CloudWatch エージェントによってインポートされたプロセスメトリクスの表示

プロセスメトリクスを CloudWatch にインポートした後、これらのメトリクスを時系列グラフとして表示し、これらのメトリクスをモニターリングし、指定したしきい値に違反した場合に通知できるアラームを作成できます。以下の手順では、プロセスメトリクスを時系列グラフとして表示する方法を示します。アラームの設定の詳細については、「Amazon CloudWatch でのアラームの使用」を参照してください。

CloudWatch コンソールでプロセスメトリクスを表示するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで [Metrics (メトリクス)] を選択します。
3. エージェントによって収集されたメトリクスの名前空間を選択します。デフォルトでは、これは CWAgent ですが、CloudWatch エージェント設定ファイルで別の名前空間を指定している場合があります。
4. メトリクスのディメンション (例: インスタンス別メトリクス) を選択します。
5. [All metrics] タブには、名前空間内のそのディメンションのメトリクスがすべて表示されます。以下の操作を行うことができます。

   a. メトリクスをグラフ表示するには、メトリクスの横にあるチェックボックスを選択します。すべてのメトリクスを選択するには、テーブルの見出し行にあるチェックボックスを選択します。

   b. テーブルを並べ替えるには、列見出しを使用します。

   c. リソースでフィルタするには、リソース ID を選択し、[Add to search] を選択します。

   d. メトリクスでフィルタするには、メトリクスの名前を選択し、[Add to search] を選択します。
6. (オプション) このグラフを CloudWatch ダッシュボードに追加するには、[Actions] (アクション)、[Add to dashboard] (ダッシュボードに追加) の順に選択します。

StatsD を使用してカスタムメトリクスを取得する

CloudWatch エージェントで StatsD プロトコルを使用することで、アプリケーションやサービスから追加のカスタムメトリクスを取得できます。StatsDは、さまざまなアプリケーションからメトリクスを収集できる一般的なオープンソースソリューションです。StatsD は、独自のメトリクスを計測する場合に特に便利です。CloudWatch エージェントと StatsD を一緒に使用する例については、「Amazon CloudWatch エージェントを使用してカスタムアプリケーションメトリクスをより効率的にモニターリングする方法」を参照してください。

StatsD は、Linux サーバーと、Windows Server を実行するサーバーの両方でサポートされます。CloudWatch では、次の StatsD 形式がサポートされています。

```
MetricName:value|type|@sample_rate|#tag1:
  value,tag1...
```

- MetricName – コロン、バー、# 文字、または @ 文字を含まない文字列。
- value – これは整数または浮動小数点のいずれかです。
- type – カウンターには c、ゲージには g、タイマーには ms、ヒストグラムには h、セットには s を指定します。
- sample_rate – (オプション) 0 から 1 の間の浮動小数点数 (0 と 1 を含む)。カウンター、ヒストグラム、およびタイマーメトリクスに対してのみ使用します。デフォルト値は 1 (時間の 100% サンプリング) です。
- tags – (オプション) タグのカンマ区切りリスト。StatsD タグは CloudWatch のディメンションと似ています。キー/値タグにはコロンを使用します、例 env:prod。

この形式に従う任意の StatsD クライアントを使用して、CloudWatch エージェントにメトリクスを送信することができます。一部の利用可能な StatsD クライアントの詳細については、「GitHub の StatsD クライアントページ」を参照してください。

これらのカスタムメトリクスを収集するには、"statsd": {} 行をエージェント設定ファイルの metrics_collected セクションに追加します。この行を手動で追加することができます。設定ファイルの作成にウィザードを使用する場合は、自動的に設定されます。詳細については、「CloudWatch エージェント設定ファイルを作成する」を参照してください。

StatsD のデフォルト設定は、ほとんどのユーザーに有効です。また、必要に応じてエージェント設定ファイルの [statsd] セクションに追加できるオプションフィールドがあります。

- service_address – CloudWatch エージェントがリッスンする必要のあるサービスアドレス。形式は *ip*:*port* です。IP アドレスを省略すると、エージェントは使用可能なすべてのインターフェイスをリッスンします。UDP 形式のみがサポートされているため、UDP プレフィックスを指定する必要はありません。

  デフォルト値は :8125 です。
- metrics_collection_interval – StatsD プラグインが実行され、メトリクスを収集する秒単位の頻度。デフォルト値は 10 秒です。範囲は 1 ~ 172,000 です。
- metrics_aggregation_interval – CloudWatch がメトリクスを 1 つのデータポイントに集約する秒単位の頻度。デフォルト値は 60 秒です。

  例えば、metrics_collection_interval が 10 で metrics_aggregation_interval が 60 の場合、CloudWatch は 10 秒ごとにデータを収集します。1 分ごとに、その 1 分間に読み取った 6 つのデータ読み取り値が 1 つのデータポイントに集約され、それが CloudWatch に送信されます。

範囲は 0 ~ 172,000 です。metrics_aggregation_interval を 0 に設定すると、StatsD メトリクスの集計が無効になります。

- allowed_pending_messages — キューに入れることができる UDP メッセージの数です。キューがフルになると、StatsD サーバーはパケットのドロップを開始します。デフォルト値は 10000 です。
- drop_original_metrics - オプション。metrics セクションの aggregation_dimensions フィールドを使用してメトリクスを集計結果にロールアップしている場合、デフォルトでは、エージェントは集計メトリクスと、ディメンションの値ごとに分離された元のメトリクスの両方を送信します。元のメトリクスを CloudWatch に送信したくない場合は、メトリクスのリストを使用してこのパラメータを指定できます。このパラメータとともに指定されたメトリクスには、CloudWatch にレポートされるディメンションごとのメトリクスがありません。代わりに、集計されたメトリクスのみがレポートされます。これにより、エージェントが収集するメトリクスの数が減り、コストが削減されます。

次に示すのは、デフォルトポートとカスタム収集および集約間隔を使用した、エージェント設定ファイルの [statsd] セクションの例です。

```
{
   "metrics":{
      "metrics_collected":{
         "statsd":{
            "service_address":":8125",
            "metrics_collection_interval":60,
            "metrics_aggregation_interval":300
         }
      }
   }
}
```

CloudWatch エージェントによってインポートされた StatsD メトリクスの表示

StatsD メトリクスを CloudWatch にインポートした後、これらのメトリクスを時系列グラフとして表示し、これらのメトリクスをモニターリングし、指定したしきい値に違反した場合に通知できるアラームを作成できます。以下の手順では、StatsD メトリクスを時系列グラフとして表示する方法を示します。アラームの設定の詳細については、「Amazon CloudWatch でのアラームの使用」を参照してください。

CloudWatch コンソールで StatsD メトリクスを表示する

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで [Metrics (メトリクス)] を選択します。
3. エージェントによって収集されたメトリクスの名前空間を選択します。デフォルトでは、これは CWAgent ですが、CloudWatch エージェント設定ファイルで別の名前空間を指定している場合があります。
4. メトリクスのディメンション (例: インスタンス別メトリクス) を選択します。
5. [All metrics] タブには、名前空間内のそのディメンションのメトリクスがすべて表示されます。以下の操作を行うことができます。
   a. メトリクスをグラフ表示するには、メトリクスの横にあるチェックボックスを選択します。すべてのメトリクスを選択するには、テーブルの見出し行にあるチェックボックスを選択します。
   b. テーブルを並べ替えるには、列見出しを使用します。
   c. リソースでフィルタするには、リソース ID を選択し、[Add to search] を選択します。
   d. メトリクスでフィルタするには、メトリクスの名前を選択し、[Add to search] を選択します。
6. (オプション) このグラフを CloudWatch ダッシュボードに追加するには、[Actions] (アクション)、[Add to dashboard] (ダッシュボードに追加) の順に選択します。

collectd を使用してカスタムメトリクスを取得する

CloudWatch エージェントで collectd プロトコルを使用することで、アプリケーションやサービスから追加のメトリクスを取得できます。collectd プロトコルは Linux サーバーでのみサポートされています。collectd は、さまざまなアプリケーションのシステム統計を収集できるプラグインを備えた、一般的なオープンソースソリューションです。CloudWatch エージェントが既に収集できるシステムメトリクスと collectd からの追加のメトリクスを組み合わせることで、システムやアプリケーションのモニターリング、分析、トラブルシューティングを強化できます。collectd の詳細については、「collectd - The system statistics collection daemon」を参照してください。

collectd ソフトウェアを使用して CloudWatch エージェントにメトリクスを送信します。collectd メトリクスでは、CloudWatch エージェントはサーバーとして機能するのに対し、collectd プラグインはクライアントとして機能します。

collectd ソフトウェアは、すべてのサーバーに自動的にインストールされるわけではありません。Amazon Linux 2 を実行するサーバーで、次の手順に従って collectd をインストールします。

```
sudo amazon-linux-extras install collectd
```

他のシステムに collectd をインストールする方法については、collectd の ダウンロードページを参照してください。

これらのカスタムメトリクスを収集するには、["collectd": {}] 行をエージェント設定ファイルの [metrics_collected] セクションに追加します。この行を手動で追加することができます。設定ファイルの作成にウィザードを使用する場合は、自動的に設定されます。詳細については、「CloudWatch エージェント設定ファイルを作成する」を参照してください。

オプションパラメータも使用可能です。collectd を使用していて [collectd_auth_file] として `/etc/collectd/auth_file` を使用しない場合は、これらのオプションのいくつかを設定する必要があります。

- service_address: CloudWatch エージェントが聞く必要のあるサービスアドレス。形式は "`udp://`*`ip`*`:`*`port`* です。デフォルト: `udp://127.0.0.1:25826`。
- name_prefix: 各 collectd メトリクスの名前の先頭にアタッチするプレフィックスです。デフォルト: `collectd_`。最大長は 255 文字です。
- collectd_security_level: ネットワーク通信のセキュリティレベルを設定します。デフォルトは [encrypt] です。

  [encrypt] は、暗号化されたデータのみが受け入れられることを指定します。[sign] は、署名され暗号化されたデータのみが受け入れられることを指定します。[none] は、すべてのデータが受け入れられることを指定します。[collectd_auth_file] の値を指定すると、暗号化されたデータは可能であれば復号されます。

  詳細については、collectd Wiki の「クライアント設定」および「可能なインタラクション」を参照してください。
- collectd_auth_file: ユーザー名がパスワードにマッピングされるファイルを設定します。これらのパスワードは、署名の検証と暗号化されたネットワークパケットの復号に使用されます。指定された場合、署名されたデータが検証され、暗号化されたパケットが復号されます。それ以外の場合は、署名されたデータは署名をチェックすることなく受け入れられ、暗号化されたデータを復号することはできません。

  デフォルト: `/etc/collectd/auth_file`。

[collectd_security_level] が [none] に設定されている場合、これはオプションです。[collectd_security_level] を `encrypt` または [sign] に設定した場合は、[collectd_auth_file] を指定する必要があります。

auth ファイルの形式は、各行はユーザー名とそれに続くコロン、さらに任意の数のスペースとそれに続くパスワードの構成です。次に例を示します。

```
user1: user1_password

user2: user2_password
```

- collectd_typesdb: データセットの説明を含む 1 つ以上のファイルのリストです。リストにエントリが 1 つしかない場合でも、リストは角括弧で囲まれている必要があります。リストの各エントリは二重引用符で囲む必要があります。複数のエントリがある場合は、カンマで区切ります Linux サーバーのデフォルトは `["/usr/share/collectd/types.db"]` です。macOS コンピュータのデフォルトは、collectd のバージョンによって異なります。例えば、`["/usr/local/Cellar/collectd/5.12.0/share/collectd/types.db"]` と指定します。

  詳細については、「https://www.collectd.org/documentation/manpages/types.db.html」を参照してください。

- metrics_aggregation_interval: CloudWatch がメトリクスを 1 つのデータポイントに集約する秒単位の頻度。デフォルト値は 60 秒です。範囲は 0 ~ 172,000 です。0 に設定すると、collectd メトリクスの集計が無効になります。

エージェント設定ファイルの collectd セクションの例を次に示します。

```
{
   "metrics":{
      "metrics_collected":{
         "collectd":{
            "name_prefix":"My_collectd_metrics_",
            "metrics_aggregation_interval":120
         }
      }
   }
}
```

CloudWatch エージェントによってインポートされた collectd メトリクスの表示

collectd メトリクスを CloudWatch にインポートした後、これらのメトリクスを時系列グラフとして表示し、これらのメトリクスをモニターリングし、指定したしきい値に違反した場合に通知できるアラームを作成できます。以下の手順では、collectd メトリクスを時系列グラフとして表示する方法を示します。アラームの設定の詳細については、「 Amazon CloudWatch でのアラームの使用」を参照してください。

CloudWatch コンソールで collectd メトリクスを表示するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで [Metrics (メトリクス)] を選択します。
3. エージェントによって収集されたメトリクスの名前空間を選択します。デフォルトでは、これは CWAgent ですが、CloudWatch エージェント設定ファイルで別の名前空間を指定している場合があります。
4. メトリクスのディメンション (例: インスタンス別メトリクス) を選択します。
5. [All metrics] タブには、名前空間内のそのディメンションのメトリクスがすべて表示されます。以下の操作を行うことができます。

   a. メトリクスをグラフ表示するには、メトリクスの横にあるチェックボックスを選択します。すべてのメトリクスを選択するには、テーブルの見出し行にあるチェックボックスを選択します。

   b. テーブルを並べ替えるには、列見出しを使用します。

   c. リソースでフィルタするには、リソース ID を選択し、[Add to search] を選択します。

   d. メトリクスでフィルタするには、メトリクスの名前を選択し、[Add to search] を選択します。
6. (オプション) このグラフを CloudWatch ダッシュボードに追加するには、[Actions] (アクション)、[Add to dashboard] (ダッシュボードに追加) の順に選択します。

Amazon EC2 インスタンスでの Prometheus メトリクスコレクションのセットアップと設定

次のセクションでは、EC2 インスタンスに Prometheus モニターリングを使用して CloudWatch エージェントをインストールする方法と、追加のターゲットをスクレイプするようにエージェントを設定する方法について説明します。また、Prometheus モニターリングでのテストに使用するサンプルワークロードを設定するためのオチュートリアルも提供します。

CloudWatch エージェントがサポートするオペレーティングシステムについては、「CloudWatch エージェントを使用してメトリクス、ログ、トレースを収集する」を参照してください。

VPC セキュリティグループの要件

VPC を使用している場合は、次の要件が適用されます。

- Prometheus ワークロードのセキュリティグループの受信ルールでは、Prometheus のメトリクスをプライベート IP でスクレイピングするために、CloudWatch エージェントへの Prometheus ポートを開く必要があります。
- CloudWatch エージェントのセキュリティグループの出力ルールでは、CloudWatch エージェントがプライベート IP によって Prometheus ワークロードのポートに接続できるようにする必要があります。

トピック

- ステップ 1: CloudWatch エージェントをインストールする
- ステップ 2: Prometheus ソースとインポートメトリクスをスクレイプする
- 例: Prometheus メトリクステスト用に Java/JMX サンプルワークロードを設定する

ステップ 1: CloudWatch エージェントをインストールする

最初のステップは、EC2 インスタンスに CloudWatch エージェントをインストールすることです。手順については、「CloudWatch エージェントのインストール」を参照してください。

ステップ 2: Prometheus ソースとインポートメトリクスをスクレイプする

Prometheus モニターリングを使用した CloudWatch エージェントは、Prometheus メトリクスをスクレイプするために 2 つの設定が必要です。1 つは標準の Prometheus 設定用で、Prometheus ドキュメントの「<scrape_config>」に記載されています。もう 1 つは CloudWatch エージェント設定用です。

Prometheus スクレイプ設定

この CloudWatch エージェントは、Prometheus のドキュメントの「<scrape_config>」に記載されているように、標準の Prometheus スクレイプ設定をサポートしています。このセクションを編集して、このファイルに既に含まれている設定を更新したり、Prometheus スクレイピングターゲットを追加したりできます。サンプル設定ファイルに次のグローバル設定行が含まれています。

```
PS C:\ProgramData\Amazon\AmazonCloudWatchAgent> cat prometheus.yaml
global:
  scrape_interval: 1m
  scrape_timeout: 10s
scrape_configs:
- job_name: MY_JOB
  sample_limit: 10000
  file_sd_configs:
    - files: ["C:\\ProgramData\\Amazon\\AmazonCloudWatchAgent\\prometheus_sd_1.yaml",
 "C:\\ProgramData\\Amazon\\AmazonCloudWatchAgent\\prometheus_sd_2.yaml"]
```

global セクションでは、すべての設定コンテキストで有効なパラメータを指定します。また、これらは他の設定セクションのデフォルトとしても機能します。これには以下のパラメータが含まれています。

- scrape_interval – ターゲットをスクレイプする頻度を定義します。
- scrape_timeout – スクレイプリクエストがタイムアウトするまでの待機時間を定義します。

scrape_configs セクションでは、ターゲットとパラメータのセットを指定し、それらをスクレイプする方法を定義します。これには以下のパラメータが含まれています。

- job_name – デフォルトでスクレイプされたメトリクスに割り当てられたジョブ名。
- sample_limit – スクレイプごとに受け入れられるスクレイプされたサンプルの数の制限。
- file_sd_configs – ファイルサービスの検出設定のリスト。これは、ゼロ個以上の静的設定のリストを含むファイルのセットを読み取ります。file_sd_configs セクションには、ターゲットグループの抽出元となるファイルのパターンを定義する files パラメータが含まれています。

CloudWatch エージェントは、次のサービス検出設定タイプをサポートします。

static_config ターゲットのリストと、それらに共通のラベルセットを指定できます。これは、スクレイプ設定で静的ターゲットを指定するための標準的な方法です。

以下に、ローカルホストから Prometheus メトリクスをスクレーピングするための静的設定の例を示します。エージェントが実行されているサーバーに対して Prometheus ポートが開いている場合は、メトリクスを他のサーバーからスクレイプすることもできます。

```
PS C:\ProgramData\Amazon\AmazonCloudWatchAgent> cat prometheus_sd_1.yaml
- targets:
```

```
    - 127.0.0.1:9404
  labels:
    key1: value1
    key2: value2
```

この例には、以下のキー要素が含まれています。

- `targets` – 静的設定によってスクレイプされたターゲット。
- `labels` – ターゲットからスクレイプされたすべてのメトリクスに割り当てられたラベル。

`ec2_sd_config` Amazon EC2 インスタンスからスクレイプターゲットを取得できます。EC2 インスタンスのリストから Prometheus メトリクスをスクレイプするサンプル `ec2_sd_config` を次に示します。これらのインスタンスの Prometheus ポートは、CloudWatch エージェントが実行されているサーバーに対して開く必要があります。CloudWatch エージェントが実行される EC2 インスタンスの IAM ロールには、`ec2:DescribeInstance` アクセス許可を含める必要があります。例えば、管理ポリシー AmazonEC2ReadOnlyAccess を CloudWatch エージェントを実行するインスタンスにアタッチできます。

```
PS C:\ProgramData\Amazon\AmazonCloudWatchAgent> cat prometheus.yaml
global:
  scrape_interval: 1m
  scrape_timeout: 10s
scrape_configs:
  - job_name: MY_JOB
    sample_limit: 10000
    ec2_sd_configs:
      - region: us-east-1
        port: 9404
        filters:
          - name: instance-id
            values:
              - i-98765432109876543
              - i-12345678901234567
```

この例には、以下のキー要素が含まれています。

- `region` — ターゲット EC2 インスタンスがある AWS リージョン。これを空白のままにすると、インスタンスメタデータのリージョンが使用されます。
- `port` – メトリクスをスクレイプするポート。

- `filters` – インスタンスリストのフィルタリングに使用するオプションのフィルター。この例では、EC2 インスタンス ID に基づいてフィルタリングします。フィルタリングできるその他の条件については、「DescribeInstances」を参照してください。

Prometheus の CloudWatch エージェント設定

CloudWatch エージェント設定ファイルには、`prometheus` と `logs` の両方の `metrics_collected` セクションが含まれています。それには次のパラメータが含まれます。

- cluster_name – ログイベントのラベルとして追加されるクラスター名を指定します。このフィールドはオプションです。
- log_group_name – スクレイプされた Prometheus メトリクスのロググループ名を指定します。
- prometheus_config_path – Prometheus スクレイプ設定ファイルパスを指定します。
- emf_processor – 埋め込みメトリクス形式のプロセッサ設定を指定します。埋め込みメトリックフォーマットの詳細については、「ログ内へのメトリクスの埋め込み」を参照してください。

  `emf_processor` セクションには、次のパラメータを含めることができます。

  - metric_declaration_dedup – trueに設定すると、埋め込まれたメトリクスフォーマットメトリクスの重複除外機能が有効になります。
  - metric_namespace – 出力される CloudWatch メトリクスのメトリクス名前空間を指定します。
  - metric_unit – メートルの name:metric 単位マップを指定します。サポートされているメトリクス単位の詳細については、「MetricDatum」を参照してください。
  - metric_declaration – 生成されるメトリクス形式が埋め込まれたログの配列を指定するセクションです。CloudWatch エージェントがインポートする各 Prometheus ソースには、デフォルトで `metric_declaration` セクションがあります。これらの各セクションには、次のフィールドが含まれています。
    - `source_labels` は、`label_matcher` 行によってチェックされるラベルの値を指定します。
    - `label_matcher` は、 `source_labels` に表示されているラベルの値をチェックする正規表現です。一致するメトリクスは、CloudWatch に送信される埋め込みメトリクス形式に含めることができます。
    - `metric_selectors` は、収集され、CloudWatch に送信されるメトリクスを指定する正規表現です。
    - `dimensions` は、選択した各メトリクスの CloudWatch ディメンションとして使用されるラベルのリストです。

以下に、Prometheus の CloudWatch エージェントの設定例を示します。

```
{
   "logs":{
      "metrics_collected":{
         "prometheus":{
            "cluster_name":"prometheus-cluster",
            "log_group_name":"Prometheus",
            "prometheus_config_path":"C:\\ProgramData\\Amazon\\AmazonCloudWatchAgent\
\prometheus.yaml",
            "emf_processor":{
               "metric_declaration_dedup":true,
               "metric_namespace":"CWAgent-Prometheus",
               "metric_unit":{
                  "jvm_threads_current": "Count",
                  "jvm_gc_collection_seconds_sum": "Milliseconds"
               },
               "metric_declaration":[
                  {
                     "source_labels":[
                        "job", "key2"
                     ],
                     "label_matcher":"MY_JOB;^value2",
                     "dimensions":[
                        [
                           "key1", "key2"
                        ],
                        [
                           "key2"
                        ]
                     ],
                     "metric_selectors":[
                        "^jvm_threads_current$",
                        "^jvm_gc_collection_seconds_sum$"
                     ]
                  }
               ]
            }
         }
      }
   }
}
```

前の例では、次の条件が満たされた場合にログイベントとして送信される埋め込みメトリクス形式セクションを設定します。

- job ラベルの値は MY_JOB です
- key2 ラベルの値は value2 です
- Prometheus メトリクス jvm_threads_current および jvm_gc_collection_seconds_sum には、job ラベルおよび key2 ラベルの両方が含まれます。

送信されるログイベントには、次の強調表示されたセクションが含まれます。

```
{
    "CloudWatchMetrics": [
        {
            "Metrics": [
                {
                    "Unit": "Count",
                    "Name": "jvm_threads_current"
                },
                {
                    "Unit": "Milliseconds",
                    "Name": "jvm_gc_collection_seconds_sum"
                }
            ],
            "Dimensions": [
                [
                    "key1",
                    "key2"
                ],
                [
                    "key2"
                ]
            ],
            "Namespace": "CWAgent-Prometheus"
        }
    ],
    "ClusterName": "prometheus-cluster",
    "InstanceId": "i-0e45bd06f196096c8",
    "Timestamp": "1607966368109",
    "Version": "0",
    "host": "EC2AMAZ-PDDOIUM",
    "instance": "127.0.0.1:9404",
```

```
    "jvm_threads_current": 2,
    "jvm_gc_collection_seconds_sum": 0.006000000000000002,
    "prom_metric_type": "gauge",
    ...
}
```

例: Prometheus メトリクステスト用に Java/JMX サンプルワークロードを設定する

JMX Exporter は、Prometheus メトリクスとして JMX mBeans をスクレイプおよび公開できる公式の Prometheus エクスポーターです。詳細については、prometheus/jmx_exporter を参照してください。

CloudWatch エージェントは、EC2 インスタンスの JMX Exporter から Java 仮想マシン (JVM)、Hjava、Tomcat (Catalina) から事前定義された Prometheus メトリクスを収集できます。

ステップ 1: CloudWatch エージェントをインストールする

最初のステップは、EC2 インスタンスに CloudWatch エージェントをインストールすることです。手順については、「CloudWatch エージェントのインストール」を参照してください。

ステップ 2: Java/JMX ワークロードを開始する

次のステップは、Java/JMX ワークロードを開始することです。

まず、次の場所から最新の JMX Exporter の jar ファイルをダウンロードします: prometheus/jmx_exporter。

サンプルアプリケーションに jar を使用する

次のセクションのコマンド例では、`SampleJavaApplication-1.0-SNAPSHOT.jar` を jar ファイルとして使用します。コマンドのこれらの部分をアプリケーションの jar に置き換えます。

JMX Exporter 設定ファイルを準備する

`config.yaml` ファイルは JMX Exporter 設定ファイルです。詳細については、JMX Exporter ドキュメントの「設定」を参照してください。

Java と Tomcat の設定のサンプルを次に示します。

```
---
lowercaseOutputName: true
lowercaseOutputLabelNames: true

rules:
```

```
- pattern: 'java.lang<type=OperatingSystem><>(FreePhysicalMemorySize|
TotalPhysicalMemorySize|FreeSwapSpaceSize|TotalSwapSpaceSize|SystemCpuLoad|
ProcessCpuLoad|OpenFileDescriptorCount|AvailableProcessors)'
  name: java_lang_OperatingSystem_$1
  type: GAUGE

- pattern: 'java.lang<type=Threading><>(TotalStartedThreadCount|ThreadCount)'
  name: java_lang_threading_$1
  type: GAUGE

- pattern: 'Catalina<type=GlobalRequestProcessor, name=\"(\w+-\w+)-(\d+)\"><>(\w+)'
  name: catalina_globalrequestprocessor_$3_total
  labels:
    port: "$2"
    protocol: "$1"
  help: Catalina global $3
  type: COUNTER

- pattern: 'Catalina<j2eeType=Servlet, WebModule=//([-a-zA-Z0-9+&@#/%?=~_|!:.,;]*[-
a-zA-Z0-9+&@#/%=~_|]), name=([-a-zA-Z0-9+/$%~_-|!.]*), J2EEApplication=none,
 J2EEServer=none><>(requestCount|maxTime|processingTime|errorCount)'
  name: catalina_servlet_$3_total
  labels:
    module: "$1"
    servlet: "$2"
  help: Catalina servlet $3 total
  type: COUNTER

- pattern: 'Catalina<type=ThreadPool, name="(\w+-\w+)-(\d+)"><>(currentThreadCount|
currentThreadsBusy|keepAliveCount|pollerThreadCount|connectionCount)'
  name: catalina_threadpool_$3
  labels:
    port: "$2"
    protocol: "$1"
  help: Catalina threadpool $3
  type: GAUGE

- pattern: 'Catalina<type=Manager, host=([-a-zA-Z0-9+&@#/%?=~_|!:.,;]*[-a-zA-
Z0-9+&@#/%=~_|]), context=([-a-zA-Z0-9+/$%~_-|!.]*)><>(processingTime|sessionCounter|
rejectedSessions|expiredSessions)'
  name: catalina_session_$3_total
  labels:
    context: "$2"
    host: "$1"
```

```
  help: Catalina session $3 total
  type: COUNTER

- pattern: ".*"
```

Prometheus エクスポーターで Java アプリケーションを起動する

サンプルアプリケーションを起動する これにより、Prometheus メトリクスをポート 9404 に出力します。エントリポイント com.gubupt.sample.app.App は、必ずサンプル Java アプリケーションの正しい情報に置き換えてください。

Linux では、以下のコマンドを入力します。

```
$ nohup java -javaagent:./jmx_prometheus_javaagent-0.14.0.jar=9404:./config.yaml -cp
  ./SampleJavaApplication-1.0-SNAPSHOT.jar com.gubupt.sample.app.App &
```

Windows では、以下のコマンドを入力します。

```
PS C:\> java -javaagent:.\jmx_prometheus_javaagent-0.14.0.jar=9404:.\config.yaml -cp  .
\SampleJavaApplication-1.0-SNAPSHOT.jar com.gubupt.sample.app.App
```

Prometheus メトリクスの放出を確認する

Prometheus メトリクスが出力されていることを確認します。

Linux では、以下のコマンドを入力します。

```
$ curl localhost:9404
```

Windows では、以下のコマンドを入力します。

```
PS C:\> curl  http://localhost:9404
```

Linux での出力例:

```
StatusCode        : 200
StatusDescription : OK
Content           : # HELP jvm_classes_loaded The number of classes that are currently
 loaded in the JVM
                    # TYPE jvm_classes_loaded gauge
                    jvm_classes_loaded 2526.0
```

```
                    # HELP jvm_classes_loaded_total The total number of class...
RawContent        : HTTP/1.1 200 OK
                    Content-Length: 71908
                    Content-Type: text/plain; version=0.0.4; charset=utf-8
                    Date: Fri, 18 Dec 2020 16:38:10 GMT

                    # HELP jvm_classes_loaded The number of classes that are
 currentl...
Forms             : {}
Headers           : {[Content-Length, 71908], [Content-Type, text/plain; version=0.0.4;
 charset=utf-8], [Date, Fri, 18
                    Dec 2020 16:38:10 GMT]}
Images            : {}
InputFields       : {}
Links             : {}
ParsedHtml        : System.__ComObject
RawContentLength  : 71908
```

ステップ 3: Prometheus メトリクスをスクレイプするように CloudWatch エージェントを設定する

次に、CloudWatch エージェント設定ファイルで Prometheus スクレイプ設定をセットアップします。

Java/JMX の例の Prometheus スクレイプ設定を行うには

1. `file_sd_config` および `static_config` の設定をセットアップします。

   Linux では、以下のコマンドを入力します。

   ```
   $ cat /opt/aws/amazon-cloudwatch-agent/var/prometheus.yaml
   global:
     scrape_interval: 1m
     scrape_timeout: 10s
   scrape_configs:
     - job_name: jmx
       sample_limit: 10000
       file_sd_configs:
         - files: [ "/opt/aws/amazon-cloudwatch-agent/var/prometheus_file_sd.yaml" ]
   ```

   Windows では、以下のコマンドを入力します。

   ```
   PS C:\ProgramData\Amazon\AmazonCloudWatchAgent> cat prometheus.yaml
   ```

```
global:
  scrape_interval: 1m
  scrape_timeout: 10s
scrape_configs:
  - job_name: jmx
    sample_limit: 10000
    file_sd_configs:
      - files: [ "C:\\ProgramData\\Amazon\\AmazonCloudWatchAgent\
\prometheus_file_sd.yaml" ]
```

2. スクレイプターゲットの設定を行います。

   Linux では、以下のコマンドを入力します。

```
$ cat /opt/aws/amazon-cloudwatch-agent/var/prometheus_file_sd.yaml
- targets:
  - 127.0.0.1:9404
  labels:
    application: sample_java_app
    os: linux
```

   Windows では、以下のコマンドを入力します。

```
PS C:\ProgramData\Amazon\AmazonCloudWatchAgent> cat prometheus_file_sd.yaml
- targets:
  - 127.0.0.1:9404
  labels:
    application: sample_java_app
    os: windows
```

3. ec2_sc_config によって Prometheus スクレイプ設定を行います。*your-ec2-instance-id* を正しい EC2 インスタンス ID に置き換えます。

   Linux では、以下のコマンドを入力します。

```
$ cat .\prometheus.yaml
global:
  scrape_interval: 1m
  scrape_timeout: 10s
scrape_configs:
  - job_name: jmx
    sample_limit: 10000
```

```
    ec2_sd_configs:
      - region: us-east-1
        port: 9404
        filters:
          - name: instance-id
            values:
              - your-ec2-instance-id
```

Windows では、以下のコマンドを入力します。

```
PS C:\ProgramData\Amazon\AmazonCloudWatchAgent> cat prometheus_file_sd.yaml
- targets:
  - 127.0.0.1:9404
  labels:
    application: sample_java_app
    os: windows
```

4. CloudWatch エージェント設定を行います。まず、正しいディレクトリに移動します。Linux では、それは /opt/aws/amazon-cloudwatch-agent/var/cwagent-config.json です。Windows では、それは C:\ProgramData\Amazon\AmazonCloudWatchAgent\cwagent-config.json です。

   次に、Java/JHX Prometheus メトリクスが定義されている設定例を示します。必ず *path-to-Prometheus-Scrape-Configuration-file* へのパスを正しいパスに置き換えてください。

```
{
  "agent": {
    "region": "us-east-1"
  },
  "logs": {
    "metrics_collected": {
      "prometheus": {
        "cluster_name": "my-cluster",
        "log_group_name": "prometheus-test",
        "prometheus_config_path": "path-to-Prometheus-Scrape-Configuration-file",
        "emf_processor": {
          "metric_declaration_dedup": true,
          "metric_namespace": "PrometheusTest",
          "metric_unit":{
            "jvm_threads_current": "Count",
            "jvm_classes_loaded": "Count",
            "java_lang_operatingsystem_freephysicalmemorysize": "Bytes",
```

```
            "catalina_manager_activesessions": "Count",
            "jvm_gc_collection_seconds_sum": "Seconds",
            "catalina_globalrequestprocessor_bytesreceived": "Bytes",
            "jvm_memory_bytes_used": "Bytes",
            "jvm_memory_pool_bytes_used": "Bytes"
          },
          "metric_declaration": [
            {
              "source_labels": ["job"],
              "label_matcher": "^jmx$",
              "dimensions": [["instance"]],
              "metric_selectors": [
                "^jvm_threads_current$",
                "^jvm_classes_loaded$",
                "^java_lang_operatingsystem_freephysicalmemorysize$",
                "^catalina_manager_activesessions$",
                "^jvm_gc_collection_seconds_sum$",
                "^catalina_globalrequestprocessor_bytesreceived$"
              ]
            },
            {
              "source_labels": ["job"],
              "label_matcher": "^jmx$",
              "dimensions": [["area"]],
              "metric_selectors": [
                "^jvm_memory_bytes_used$"
              ]
            },
            {
              "source_labels": ["job"],
              "label_matcher": "^jmx$",
              "dimensions": [["pool"]],
              "metric_selectors": [
                "^jvm_memory_pool_bytes_used$"
              ]
            }
          ]
        }
      }
    },
    "force_flush_interval": 5
  }
}
```

5. 次のコマンドのいずれかを入力して、CloudWatch エージェントを再起動します。

   Linux では、以下のコマンドを入力します。

   ```
   sudo /opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-ctl -a fetch-
   config -m ec2 -s -c file:/opt/aws/amazon-cloudwatch-agent/var/cwagent-config.json
   ```

   Windows では、以下のコマンドを入力します。

   ```
   & "C:\Program Files\Amazon\AmazonCloudWatchAgent\amazon-cloudwatch-agent-ctl.ps1"
    -a fetch-config -m ec2 -s -c file:C:\ProgramData\Amazon\AmazonCloudWatchAgent
   \cwagent-config.json
   ```

Prometheus のメトリクスとログの表示

これで、収集されている Java/JMX メトリクスを表示できるようになりました。

サンプル Java/JMX ワークロードのメトリクスを表示するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. クラスターが実行されているリージョンにおいて、左のナビゲーションペインで [Metrics] (メトリクス) を選択します。PrometheusTest 名前空間を検索して、メトリクスを確認します。
3. CloudWatch Logs イベントを表示するには、ナビゲーションペインで [Log Groups (ロググループ)] を選択します。イベントは、ロググループの prometheus-test にあります。

Amazon CloudWatch Observability EKS アドオンを使用して CloudWatch エージェントをインストールする

Amazon CloudWatch Observability EKS アドオンは、CloudWatch エージェントと Fluent-bit エージェントを Amazon EKS クラスターにインストールします。Amazon EKS 向けにオブザーバビリティが強化された Container Insights と、CloudWatch Application Signals はデフォルトで有効になっています。このアドオンを使用すると、Amazon EKS クラスターからインフラストラクチャメトリクス、アプリケーションパフォーマンステレメトリ、コンテナログを収集できます。

Amazon EKS 向けにオブザーバビリティが強化された Container Insights では、観察結果ごと Container Insights メトリクスに課金されます。保存されたメトリクスまたは取り込まれたログごとには課金されません。Application Signals の場合、アプリケーションへのインバウンドリクエスト、

アプリケーションからのアウトバウンドリクエスト、設定された各サービスレベル目標 (SLO) に基づいて、請求が行われます。インバウンドリクエストを受信するたびにアプリケーションシグナルが 1 つ生成され、アウトバウンドリクエストが発生するたびにアプリケーションシグナルが 1 つ生成されます。各 SLO は、測定間隔ごとにアプリケーションシグナルを 2 つ作成します。CloudWatch の料金の詳細については、Amazon CloudWatch の料金をご覧ください。

Amazon EKS アドオンは、Amazon EKS クラスターの Linux と Windows の両方のワーカーノードで Container Insights を有効にします。Windows で Container Insights を有効にするには、Amazon EKS アドオンのバージョン 1.5.0 以降を使用する必要があります。現在、Amazon EKS クラスターの Windows では、Application Signals はサポートされていません。

Amazon CloudWatch Observability EKS アドオンは、Kubernetes バージョン 1.23 以降で Amazon EKS クラスターを実行している場合にサポートされます。

また、このアドオンをインストールするときは、IAM アクセス許可を付与して、CloudWatch エージェントが CloudWatch にメトリクス、ログ、トレースを送信できるようにする必要があります。次の 2 通りの方法があります。

- ワーカーノードの IAM ロールにポリシーをアタッチします。このオプションは、テレメトリを CloudWatch に送信するアクセス許可をワーカーノードに付与します。
- エージェントポッドでサービスアカウントの IAM ロールを使用し、このロールにポリシーをアタッチします。これは Amazon EKS クラスターでのみ機能します。このオプションでは、CloudWatch は適切なエージェントポッドにのみアクセスできます。

オプション 1: ワーカーノードで IAM のアクセス許可を使用してインストールする

この方法を使用するには、まず次のコマンドを入力して IAM ポリシー CloudWatchAgentServerPolicy をワーカーノードにアタッチします。このコマンドでは、*my-worker-node-role* を Kubernetes ワーカーノードが使用する IAM ロールに置き換えます。

```
aws iam attach-role-policy \
--role-name my-worker-node-role \
--policy-arn arn:aws:iam::aws:policy/CloudWatchAgentServerPolicy
```

Amazon CloudWatch Observability EKS アドオンをインストールします。アドオンをインストールするには、AWS CLI、コンソール、AWS CloudFormation または Terraform を使用できます。

AWS CLI

AWS CLI を使用して Amazon CloudWatch Observability EKS アドオンをインストールするには

次のコマンドを入力します。*my-cluster-name* を自分のクラスター名に置き換えます。

```
aws eks create-addon --addon-name amazon-cloudwatch-observability --cluster-name my-cluster-name
```

Amazon EKS console

Amazon EKS コンソールを使用して Amazon CloudWatch Observability EKS アドオンを追加するには

1. Amazon EKS コンソール (https://console.aws.amazon.com/eks/home#/clusters) を開きます。
2. 左のナビゲーションペインで [クラスター] を選択します。
3. Amazon CloudWatch Observability EKS アドオンを設定するクラスターの名前を選択します。
4. [アドオン] タブを選択します。
5. [その他のアドオンを入手] を選択します。
6. [アドオンを選択] ページで、次の操作を行います。
   a. [Amazon EKS アドオン] セクションで、[Amazon CloudWatch Observability] チェックボックスを選択します。
   b. [Next] を選択します。
7. [選択したアドオンセッティングの設定] ページで、次の操作を行います。
   a. 使用する [バージョン] を選択します。
   b. [IAM ロールを選択] で [ノードから継承] を選択します。
   c. (オプション) [オプションの構成設定] を展開できます。[コンフリクト解決方法] で [上書きする] を選択すると、既存のアドオンでの 1 つ以上の設定が、Amazon EKS アドオンの設定で上書きされます。このオプションが有効でない状態で既存の設定との競合が発生する場合は、オペレーションが失敗します。表示されたエラーメッセージを使用して、競合をトラブルシューティングできます。このオプションを選択する前に、自分で管理する必要のある設定が、Amazon EKS アドオンにより管理されていないことを確認してください。

d. [Next] を選択します。

8. [確認と追加] ページで、[作成] を選択します。アドオンのインストールが完了した後、インストールしたアドオンが表示されます。

AWS CloudFormation

AWS CloudFormation を使用して Amazon CloudWatch Observability EKS アドオンをインストールするには

my-cluster-name を自分のクラスター名に置き換えます。詳細については、「AWS::EKS::Addon」を参照してください。

```
{
    "Resources": {
        "EKSAddOn": {
            "Type": "AWS::EKS::Addon",
            "Properties": {
                "AddonName": "amazon-cloudwatch-observability",
                "ClusterName": "my-cluster-name"
            }
        }
    }
}
```

Terraform

Terraform を使用して Amazon CloudWatch Observability EKS アドオンをインストールするには

my-cluster-name を自分のクラスター名に置き換えます。詳細については、「Resource: aws_eks_addon」を参照してください。

```
resource "aws_eks_addon" "example" {
  addon_name   = "amazon-cloudwatch-observability"
  cluster_name = "my-cluster-name"
}
```

オプション 2: IAM サービスアカウントロールを使用してインストールする

この方法を使用する前に、次の前提条件を確認してください。

- Container Insights がサポートされているいずれかの AWS リージョン に、ノードがアタッチされ機能している Amazon EKS クラスターがある。サポートされているリージョンのリストについては、「Container Insights」を参照してください。
- クラスターに kubectl がインストールされ、設定されている。詳細については、Amazon EKS ユーザーガイドの「kubectl のインストール」を参照してください。
- eksctl がインストールされている。詳細については、「Amazon EKS ユーザーガイド」の「Installing or updating eksctl」を参照してください。

IAM サービスアカウントロールを使用して Amazon CloudWatch Observability EKS アドオンをインストールするには

1. クラスターに OpenID Connect (OIDC) プロバイダーがない場合、次のコマンドを入力して作成します。詳細については、「Amazon EKS ユーザーガイド」の「Configuring a Kubernetes service account to assume an IAM role」を参照してください。

```
eksctl utils associate-iam-oidc-provider --cluster my-cluster-name --approve
```

2. 次のコマンドを入力してポリシー CloudWatchAgentServerPolicy がアタッチされた IAM ロールを作成し、OIDC を使用してそのロールを引き受けるようにエージェントのサービスアカウントを設定します。*my-cluster-name* をクラスターの名前、*my-service-account-role* をサービスアカウントを関連付けるロールの名前に置き換えます。ロールがまだ存在しない場合は、eksctl によって作成されます。

```
eksctl create iamserviceaccount \
  --name cloudwatch-agent \
  --namespace amazon-cloudwatch --cluster my-cluster-name \
  --role-name my-service-account-role \
  --attach-policy-arn arn:aws:iam::aws:policy/CloudWatchAgentServerPolicy \
  --role-only \
  --approve
```

3. 次のコマンドを入力して、アドオンをインストールします。*my-cluster-name* をクラスターの名前、*111122223333* をアカウント ID、*my-service-account-role* を前の手順で作成したIAM ロールに置き換えます。

```
aws eks create-addon --addon-name amazon-cloudwatch-observability --cluster-
name my-cluster-name --service-account-role-arn arn:aws:iam::111122223333:role/my-
service-account-role
```

(オプション) その他の設定

コンテナログの収集をオプトアウトする

デフォルトでは、アドオンは Fluent Bit を使用してすべてのポッドからコンテナログを収集し、そのログを CloudWatch Logs に送信します。収集されるログについての詳細は、「Fluent Bit の設定」を参照してください。

コンテナログの収集をオプトアウトするには、アドオンの作成時または更新時に次のオプションを渡します。

```
--configuration-values '{ "containerLogs": { "enabled": false } }'
```

NVIDIA GPU のメトリクス収集をオプトアウトする

CloudWatch エージェントのバージョン 1.300034.0 以降では、デフォルトで Container Insights が EKS ワークロードから NVIDIA GPU メトリクスを収集します。これらのメトリクスは、「NVIDIA GPU メトリクス」の表に一覧表示されています。

CloudWatch `accelerated_compute_metrics` エージェント設定ファイルのオプションを `false` に設定することで、NVIDIA GPU メトリクスの収集をオプトアウトできます。このオプションは、CloudWatch設定ファイルの `metrics_collected` セクションの `kubernetes` セクションで利用できます。以下は、オプトアウトの設定の例です。

```
{
  "agent": {
    "region": "us-east-1"
  },
  "logs": {
    "metrics_collected": {
      "emf": {
      },
      "kubernetes": {
        "enhanced_container_insights": true,
        "accelerated_compute_metrics": false
      }
    },
    "force_flush_interval": 5,
  }
}
```

カスタム CloudWatch エージェント設定を使用する

CloudWatch エージェントを使用して他のメトリクスやログやトレースを収集するには、Container Insights と CloudWatch Application Signals を有効にしたままカスタム設定を指定します。そのためには、EKS アドオンを作成または更新するときに使用できる詳細設定を開き、エージェントキーの下にある設定キー内に CloudWatch エージェント設定ファイルを埋め込みます。次に、デフォルトのまま他に設定を行わない場合のエージェント設定を示します。

> **Important**
>
> 他に構成設定を行って設定をカスタマイズした場合は、そのカスタム設定がエージェントで使用されるデフォルト設定よりも優先されます。オブザーバビリティが強化された Container Insights や CloudWatch Application Signals など、デフォルトで有効になっている機能を意図せず無効にしないように注意してください。エージェント設定をカスタマイズしなければならない場合は、以下のデフォルト設定をベースラインとして使用し、適宜必要な変更を加えることをお勧めします。

```
--configuration-values '{
  "agent": {
    "config": {
      "logs": {
        "metrics_collected": {
          "app_signals": {},
          "kubernetes": {
            "enhanced_container_insights": true
          }
        }
      },
      "traces": {
        "traces_collected": {
          "app_signals": {}
        }
      }
    }

}'
```

次の例は、Windows 上の CloudWatch エージェントのデフォルトエージェント設定を示しています。Windows 上の CloudWatch エージェントは、カスタム設定をサポートしていません。

```
{
  "logs": {
    "metrics_collected": {
      "kubernetes": {
        "enhanced_container_insights": true
      },
    }
  }
}
```

アドミッションウェブフック TLS 証明書の管理

Amazon CloudWatch Observability EKS アドオンは、Kubernetes アドミッションウェブフックを利用して、`AmazonCloudWatchAgent` および `Instrumentation` カスタムリソース (CR) リクエストのほか、オプションで CloudWatch Application Signals が有効になっている場合にはクラスター上の Kubernetes ポッドリクエストも、検証したうえで必要な変更を加えます。Kubernetes の場合、安全な通信を確保するためには、API サーバーからの信頼が設定された TLS 証明書がウェブフックに必要です。

デフォルトでは、Amazon CloudWatch Observability EKS アドオンは、API サーバーとウェブフックサーバー間の通信を保護するために、自己署名 CA とこの CA の署名入りの TLS 証明書を自動生成します。この自動生成された証明書の有効期限はデフォルトで 10 年で、有効期限は自動更新されません。また、このアドオンがアップグレードされるか再インストールされるたびに、CA バンドルと証明書が再生成されるため、有効期限がリセットされます。自動生成された証明書のデフォルトの有効期限を変更する場合は、アドオンを作成または更新するときに以下のように追加の設定を行います。*`expiry-in-days`* を目的の有効期限 (日) に置き換えてください。

```
--configuration-values '{ "admissionWebhooks": { "autoGenerateCert":
 { "expiryDays": expiry-in-days } } }'
```

安全性の高い多機能な証明機関を利用できるように、cert-manager がオプトインでサポートされています。Kubernetes で TLS 証明書を管理するのに広く採用されているソリューションであり、証明書を取得、更新、管理、使用するプロセスを簡素化できます。これにより、証明書が有効で最新の状態であることが保証されます。有効期限前の設定しておいた時刻になると、証明書が更新されます。また、AWS Certificate Manager プライベート認証機関などサポートされているさまざまな発行元から簡単に証明書を発行できます。

クラスターで TLS 証明書を管理する場合のベストプラクティスを確認すること、および本番環境では cert-manager を選択することをお勧めします。アドミッションウェブフック TLS 証明書を管理で

きるように cert-manager を有効にすることにした場合は、Amazon CloudWatch Observability EKS アドオンをインストールする前に、Amazon EKS クラスターに cert-manager をプリインストールしておく必要があることに注意してください。使用可能なインストールオプションの詳細については、cert-manager のドキュメントを参照してください。インストールを終えたら、cert-manager を使用して、アドミッションウェブフック TLS 証明書を管理することもできます。その場合は、アドオンを作成または更新するときに、以下のように追加の設定を行います。

```
--configuration-values '{ "admissionWebhooks": { "certManager": { "enabled":
 true } } }'
```

詳細設定では、ここで説明するように、デフォルトで自己署名発行者が使用されます。

Amazon EBS ボリューム ID を収集する

パフォーマンスログで Amazon EBS ボリューム ID を収集する場合は、ワーカーノードまたはサービスアカウントにアタッチされた IAM ロールに別のポリシーを追加する必要があります。以下をインラインポリシーとして追加します。詳細については、「Adding and Removing IAM Identity Permissions」を参照してください。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "ec2:DescribeVolumes"
            ],
            "Resource": "*",
            "Effect": "Allow"
        }
    ]
}
```

Amazon CloudWatch Observability EKS アドオンのトラブルシューティング

以下の情報を参考にして、Amazon CloudWatch Observability EKS アドオンに関する問題のトラブルシューティングを行います。

Amazon CloudWatch Observability EKS アドオンの更新と削除

Amazon CloudWatch Observability EKS アドオンを更新または削除する手順については、「Amazon EKS アドオンの管理」を参照してください。アドオン名には `amazon-cloudwatch-observability` を使用してください。

Amazon CloudWatch Observability EKS アドオンで使用される CloudWatch エージェントのバージョンを確認する

Amazon CloudWatch Observability EKS アドオンは、使用中の CloudWatch エージェントのバージョンなど、クラスター上の CloudWatch エージェントデーモンセットの動作を制御する `AmazonCloudWatchAgent` のカスタムリソースをインストールします。以下のコマンドを入力して、クラスターにインストールされている `AmazonCloudWatchAgent` カスタムリソースがすべて記載されたリストを取得できます。

```
kubectl get amazoncloudwatchagent -A
```

このコマンドの出力を調べれば、CloudWatch エージェントのバージョンを確認できるはずです。このほか、`amazoncloudwatchagent` リソースを記述するか、クラスターで動作しているいずれかの `cloudwatch-agent-*` ポッドを記述して、使用中のイメージを調べることもできます。

アドオン管理時の ConfigurationConflict の処理

Amazon CloudWatch Observability EKS アドオンのインストールまたは更新時に、タイプが `ConfigurationConflict` の `Health Issue` によって生じたエラー (`Conflicts found when trying to apply. Will not continue due to resolve conflicts mode` で始まる) が見られたとします。これはおそらく、CloudWatch エージェントとその関連コンポーネント (ServiceAccount、ClusterRole、ClusterRoleBinding など) がクラスターにインストールされていることが原因と考えられます。このアドオンによって CloudWatch エージェントとその関連コンポーネントがインストールされるときに内容の変更が検出されると、デフォルトではインストールまたは更新が失敗します。これによって、クラスター上にあるリソースの状態が上書きされることを防いでいます。

Amazon CloudWatch Observability EKS アドオンへのオンボード時にこのエラーが発生した場合は、クラスターにインストール済みの CloudWatch エージェントセットアップを削除した後に EKS アドオンをインストールすると良いでしょう。カスタムエージェント設定など元の CloudWatch エージェント設定に加えたカスタマイズはバックアップし、次回 Amazon CloudWatch Observability EKS アドオンをインストールまたは更新するときに、必ずそのカスタマイズを適用してくださ

い。Container Insights へのオンボーディングを目的に CloudWatch エージェントをインストール済みである場合は、「Container Insights の CloudWatch エージェントと Fluent Bit の削除」で詳細を確認してください。

その他に、アドオンでサポートされている設定オプションで OVERWRITE を指定して競合を解決することも可能です。このオプションを使用すると、クラスター上の競合を上書きしてアドオンのインストールまたは更新を継続できます。Amazon EKS コンソールを使用する場合、アドオンの作成または更新時に [オプションの構成設定] を選択すると、[競合の解決方法] が表示されます。AWS CLI を使用する場合は、コマンドに `--resolve-conflicts OVERWRITE` を指定することで、アドオンを作成または更新できます。

CloudWatch エージェントにより収集されるメトリクス

サーバーの CloudWatch エージェントをインストールすることで、サーバーからメトリクスを収集できます。Amazon EC2 インスタンスとオンプレミスサーバーの両方と、Linux、Windows Server、または macOS を実行しているコンピュータにエージェントをインストールできます。Amazon EC2 インスタンスにエージェントをインストールした場合、Amazon EC2 インスタンスでデフォルトで有効なメトリクスに加えて、メトリクスが収集されます。

インスタンスに CloudWatch エージェントをインストールする方法については、「CloudWatch エージェントを使用してメトリクス、ログ、トレースを収集する」を参照してください。

このセクションで説明するすべてのメトリクスは、CloudWatch エージェントによって直接収集されます。

Windows Server インスタンスで CloudWatch エージェントにより収集されるメトリクス

Windows Server を実行しているサーバーで、CloudWatch エージェントをインストールすると、Windows パフォーマンスモニターでカウンタに関連付けられているメトリクスを参照できます。これらのカウンタの CloudWatch メトリクス名は、オブジェクト名とカウンタ名の間にスペースを配置することによって作成されます。例えば、CloudWatch では、`% Interrupt Time` オブジェクトの `Processor` カウンタにメトリクス名 `Processor % Interrupt Time` が指定されます。Windows パフォーマンスモニターカウンタの詳細については、Microsoft Windows Server のドキュメントを参照してください。

CloudWatch エージェントにより収集されるメトリクスのデフォルトの名前空間は `CWAgent` ですが、エージェントを構成するときに別の名前空間を指定できます。

Linux および macOS インスタンスで CloudWatch エージェントにより収集されるメトリクス

次の表では、Linux サーバーおよび macOS コンピュータ上の CloudWatch エージェントで収集できるメトリクスを示しています。

メトリクス	説明
cpu_time_active	任意の容量で CPU がアクティブになっている時間の長さ。このメトリクスは、1/100 秒単位で測定されます。 単位: なし
cpu_time_guest	ゲストオペレーティングシステムで CPU が仮想 CPU を実行している時間の長さ。このメトリクスは、1/100 秒単位で測定されます。 単位: なし
cpu_time_guest_nice	優先度が低く、他のプロセスにより中断される場合がある、ゲストオペレーティングシステムの仮想 CPU を CPU で実行している時間。このメトリクスは、1/100 秒単位で測定されます。 単位: なし
cpu_time_idle	CPU がアイドル状態の時間の長さ。このメトリクスは、1/100 秒単位で測定されます。 単位: なし
cpu_time_iowait	CPU が I/O 操作の完了を待機している時間の長さ。このメトリクスは、1/100 秒単位で測定されます。 単位: なし
cpu_time_irq	CPU が中断を処理している時間の長さ。このメトリクスは、1/100 秒単位で測定されます。

メトリクス	説明
	単位: なし
cpu_time_nice	プロセスの優先度が低く、優先度の高いプロセスによって簡単に中断される場合がある、ユーザーモードになっている CPU の時間。このメトリクスは、1/100 秒単位で測定されます。 単位: なし
cpu_time_softirq	CPU がソフトウェアの中断を処理している時間の長さ。このメトリクスは、1/100 秒単位で測定されます。 単位: なし
cpu_time_steal	CPU が盗まれた時間になっている時間の長さ。これは、仮想化環境で他のオペレーティングシステムに費やされる時間です。このメトリクスは、1/100 秒単位で測定されます。 単位: なし
cpu_time_system	CPU がシステムモードになっている時間の長さ。このメトリクスは、1/100 秒単位で測定されます。 単位: なし
cpu_time_user	CPU がユーザーモードになっている時間の長さ。このメトリクスは、1/100 秒単位で測定されます。 単位: なし
cpu_usage_active	任意の容量で CPU がアクティブになっている時間の割合。 単位: パーセント

メトリクス	説明
cpu_usage_guest	ゲストオペレーティングシステムで CPU が仮想 CPU を実行している時間の割合。 単位: パーセント
cpu_usage_guest_nice	優先度が低く、他のプロセスにより中断される場合がある、ゲストオペレーティングシステムの仮想 CPU を CPU で実行している時間の割合。 単位: パーセント
cpu_usage_idle	CPU がアイドル状態の時間の割合。 単位: パーセント
cpu_usage_iowait	CPU が I/O 操作の完了を待機している時間の割合。 単位: パーセント
cpu_usage_irq	CPU が中断を処理している時間の割合。 単位: パーセント
cpu_usage_nice	プロセスの優先度が低く、優先度の高いプロセスによって簡単に中断される場合がある、ユーザーモードになっている CPU の時間の割合。 単位: パーセント
cpu_usage_softirq	CPU がソフトウェアの中断を処理している時間の割合。 単位: パーセント
cpu_usage_steal	CPU が盗まれた時間になっている時間の割合。つまり、仮想化環境で他のオペレーティングシステムに費やされる時間です。 単位: パーセント

メトリクス	説明
cpu_usage_system	CPU がシステムモードになっている時間の割合。 単位: パーセント
cpu_usage_user	CPU がユーザーモードになっている時間の割合。 単位: パーセント
disk_free	ディスクの空き容量。 単位: バイト
disk_inodes_free	ディスクで使用可能なインデックスノードの数。 単位: カウント
disk_inodes_total	ディスクで予約されているインデックスノードの合計数。 単位: カウント
disk_inodes_used	ディスクで使用されているインデックスノードの数。 単位: カウント
disk_total	使用済み容量と空き容量を含む、ディスクの合計容量。 単位: バイト
disk_used	ディスクの使用済み容量。 単位: バイト
disk_used_percent	ディスクスペース合計に対する使用済みの割合。 単位: パーセント

メトリクス	説明
diskio_iops_in_progress	デバイスドライバーに発行されたがまだ完了していない I/O リクエストの数。 単位: カウント
diskio_io_time	ディスクが I/O リクエストをキューに入れている時間の長さ。 単位: ミリ秒 このメトリクスに使用する必要がある唯一の統計は Sum です。使用しません。Average
diskio_reads	ディスク読み取り操作の回数。 単位: カウント このメトリクスに使用する必要がある唯一の統計は Sum です。使用しません。Average
diskio_read_bytes	ディスクから読み込まれたバイト数。 単位: バイト このメトリクスに使用する必要がある唯一の統計は Sum です。使用しません。Average
diskio_read_time	読み取りリクエストがディスクで待機した時間の長さ。複数の読み込みリクエストが同時に待機している場合、その分数値が増えます。たとえば、5 つのリクエストが平均 100 ミリ秒待機している場合、500 と報告されます。 単位: ミリ秒 このメトリクスに使用する必要がある唯一の統計は Sum です。使用しません。Average

メトリクス	説明
diskio_writes	ディスク書き込み操作の回数。 単位: カウント このメトリクスに使用する必要がある唯一の統計は Sum です。使用しません。Average
diskio_write_bytes	ディスクへの書き込みバイト数。 単位: バイト このメトリクスに使用する必要がある唯一の統計は Sum です。使用しません。Average
diskio_write_time	書き込みリクエストがディスクで待機した時間の長さ。複数の書き込みリクエストが同時に待機している場合、その分数値が増えます。たとえば、8 つのリクエストが平均 1000 ミリ秒待機している場合、8000 と報告されます。 単位: ミリ秒 このメトリクスに使用する必要がある唯一の統計は Sum です。使用しません。Average
ethtool_bw_in_allowance_exceeded	インバウンド集計の帯域幅がインスタンスの最大値を超えたために、キューおよび (または) ドロップされたパケットの数。 このメトリクスは、CloudWatch エージェント設定ファイルの ethtool セクションの metrics_collected サブセクションにリストされている場合にのみ収集されます。詳細については、ネットワークパフォーマンスメトリクスの収集 を参照してください。 単位: なし

メトリクス	説明
ethtool_bw_out_allowance_exceeded	アウトバウンド集計の帯域幅がインスタンスの最大値を超えたために、キューおよび (または) ドロップされたパケットの数。 このメトリクスは、CloudWatch エージェント設定ファイルの ethtool セクションの metrics_collected サブセクションにリストされている場合にのみ収集されます。詳細については、ネットワークパフォーマンスメトリクスの収集 を参照してください。 単位: なし
ethtool_conntrack_allowance_exceeded	接続トラッキングがインスタンスの最大数を超え、新しい接続を確立できなかったためにドロップされたパケットの数。これにより、インスタンスとの間で送受信されるトラフィックのパケット損失が発生する可能性があります。 このメトリクスは、CloudWatch エージェント設定ファイルの ethtool セクションの metrics_collected サブセクションにリストされている場合にのみ収集されます。詳細については、ネットワークパフォーマンスメトリクスの収集 を参照してください。 単位: なし

メトリクス	説明
ethtool_linklocal_allowance_exceeded	ローカルプロキシサービスへのトラフィックの PPS がネットワークインターフェイスの最大値を超えたためにドロップされたパケットの数。これは、DNS サービス、インスタンスメタデータサービス、および Amazon Time Sync Service へのトラフィックに影響します。 このメトリクスは、CloudWatch エージェント設定ファイルの ethtool セクションの metrics_collected サブセクションにリストされている場合にのみ収集されます。詳細については、ネットワークパフォーマンスメトリクスの収集 を参照してください。 単位: なし
ethtool_pps_allowance_exceeded	双方向 PPS がインスタンスの最大値を超えたために、キューおよび (または) ドロップされたパケットの数。 このメトリクスは、CloudWatch エージェント設定ファイルの ethtool セクションの metrics_collected サブセクションにリストされている場合にのみ収集されます。詳細については、「ネットワークパフォーマンスメトリクスの収集」を参照してください。 単位: なし
mem_active	最後のサンプル期間中に何らかの方法で使用されたメモリの量。 単位: バイト

メトリクス	説明
mem_available	すぐにプロセスに渡すことができる使用可能なメモリの量。 単位: バイト
mem_available_percent	すぐにプロセスに渡すことができる使用可能なメモリの割合。 単位: パーセント
mem_buffered	バッファに使用されているメモリの量。 単位: バイト
mem_cached	ファイルキャッシュに使用されているメモリの量。 単位: バイト
mem_free	使用されていないメモリの量。 単位: バイト
mem_inactive	最後のサンプル期間中に何らかの方法で使用されていないメモリの量 単位: バイト
mem_total	メモリの合計量。 単位: バイト
mem_used	現在使用中のメモリの量。 単位: バイト
mem_used_percent	現在使用中のメモリの割合。 単位: パーセント

メトリクス	説明
net_bytes_recv	ネットワークインターフェイスで受信されたバイトの数。 単位: バイト このメトリクスに使用する必要がある唯一の統計は Sum です。使用しません。Average
net_bytes_sent	ネットワークインターフェイスで送信されたバイトの数。 単位: バイト このメトリクスに使用する必要がある唯一の統計は Sum です。使用しません。Average
net_drop_in	このネットワークインターフェイスで受信されたパケットのうち、削除されたものの数。 単位: カウント このメトリクスに使用する必要がある唯一の統計は Sum です。使用しません。Average
net_drop_out	このネットワークインターフェイスで送信されたパケットのうち、削除されたものの数。 単位: カウント このメトリクスに使用する必要がある唯一の統計は Sum です。使用しません。Average
net_err_in	このネットワークインターフェイスによって検出された受信エラーの数。 単位: カウント このメトリクスに使用する必要がある唯一の統計は Sum です。使用しません。Average

メトリクス	説明
net_err_out	このネットワークインターフェイスによって検出された送信エラーの数。 単位: カウント このメトリクスに使用する必要がある唯一の統計は Sum です。使用しません。Average
net_packets_sent	このネットワークインターフェイスで送信されたパケットの数。 単位: カウント このメトリクスに使用する必要がある唯一の統計は Sum です。使用しません。Average
net_packets_recv	このネットワークインターフェイスで受信されたパケットの数。 単位: カウント このメトリクスに使用する必要がある唯一の統計は Sum です。使用しません。Average
netstat_tcp_close	状態のない TCP 接続の数。 単位: カウント
netstat_tcp_close_wait	クライアントからの終了リクエストを待機している TCP 接続の数。 単位: カウント
netstat_tcp_closing	クライアントからの確認付き終了リクエストを待機している TCP 接続の数。 単位: カウント

メトリクス	説明
netstat_tcp_established	確立された TCP 接続の数。 単位: カウント
netstat_tcp_fin_wait1	接続の終了プロセス時に FIN_WAIT1 状態になっている TCP 接続の数。 単位: カウント
netstat_tcp_fin_wait2	接続の終了プロセス時に FIN_WAIT2 状態になっている TCP 接続の数。 単位: カウント
netstat_tcp_last_ack	クライアントが接続終了メッセージの確認を送信するのを待機している TCP 接続の数。これは、接続が終了する直前の最後の状態です。 単位: カウント
netstat_tcp_listen	現在接続リクエストをリッスンしている TCP ポートの数。 単位: カウント
netstat_tcp_none	非アクティブなクライアントを持つ TCP 接続の数。 単位: カウント
netstat_tcp_syn_sent	接続リクエストを送信したあとに一致する接続リクエストを待機している TCP 接続の数。 単位: カウント
netstat_tcp_syn_recv	接続リクエストを送受信したあとに接続リクエスト確認を待機している TCP 接続の数。 単位: カウント

メトリクス	説明
netstat_tcp_time_wait	クライアントが接続終了リクエストの確認を受信したことが確認されるのを現在待機している TCP 接続の数。 単位: カウント
netstat_udp_socket	現在の UDP 接続の数。 単位: カウント
processes_blocked	ブロックされているプロセスの数。 単位: カウント
processes_dead	「dead」となっているプロセスの数。Linux では、X 状態コードにより示されます。 このメトリクスは macOS コンピュータでは収集されません。 単位: カウント
processes_idle	アイドル状態になっているプロセスの数 (20 秒以上スリープ状態)。FreeBSD インスタンスでのみ使用できます。 単位: カウント
processes_paging	「paging」となっているプロセスの数。Linux では、W 状態コードにより示されます。 このメトリクスは macOS コンピュータでは収集されません。 単位: カウント

メトリクス	説明
processes_running	実行されているプロセスの数。R 状態コードにより示されます。 単位: カウント
processes_sleeping	スリープ状態になっているプロセスの数。S 状態コードにより示されます。 単位: カウント
processes_stopped	停止されているプロセスの数。T 状態コードにより示されます。 単位: カウント
processes_total	インスタンス上でのプロセスの合計数。 単位: カウント
processes_total_threads	プロセスを構成するスレッドの合計数。このメトリクスは、Linux インスタンスでのみご利用いただけます。 このメトリクスは macOS コンピュータでは収集されません。 単位: カウント
processes_wait	ページングしているプロセスの数。FreeBSD インスタンスでは、W 状態コードにより示されます。このメトリクスは FreeBSD インスタンスでのみ使用でき、Linux、Windows サーバー、macOS インスタンスでは使用できません。 単位: カウント

メトリクス	説明
processes_zombies	ゾンビ状態のプロセスの数。Z 状態コードにより示されます。 単位: カウント
swap_free	使用されていないスワップスペースの量。 単位: バイト
swap_used	現在使用中のスワップスペースの量。 単位: バイト
swap_used_percent	現在使用中のスワップスペースの割合。 単位: パーセント

CloudWatch エージェントにより収集されるメモリメトリクスの定義

CloudWatch エージェントがメモリメトリクスを収集する場合、ソースはホストのメモリ管理サブシステムです。例えば、Linux カーネルは OS が管理する /proc 内のデータを公開します。メモリについては、データは /proc/meminfo にあります。

オペレーティングシステムとアーキテクチャが異なれば、プロセスが使用するリソースの計算も異なります。詳細については、次のセクションを参照してください。

各収集間隔において、各インスタンスの CloudWatch エージェントはインスタンスリソースを収集し、そのインスタンスで実行されているすべてのプロセスが使用しているリソースを計算します。この情報は CloudWatch メトリクスに報告されます。収集間隔の長さは、CloudWatch エージェント設定ファイルで設定できます。詳細については、「CloudWatch エージェント設定ファイル: Agent セクション」を参照してください。

次のリストで、CloudWatch エージェントが収集するメモリメトリクスの定義方法を説明します。

- アクティブメモリ — プロセスが使用しているメモリです。つまり、現在実行中のアプリで使用されているメモリのことです。
- 使用可能なメモリ — システムがスワップ状態にならずにプロセスに瞬時に割り当てられるメモリ (仮想メモリとも呼ばれます) です。

- バッファメモリ — さまざまな速度と優先順位で動作するハードウェアデバイスまたはプログラムプロセスによって共有されるデータ領域です。
- キャッシュされたメモリ — CPU が次に必要とする可能性のあるプログラムの操作で繰り返し使用されるプログラム手順とデータを保存します。
- 空きメモリ — まったく使用されておらず、すぐに使用できるメモリです。このメモリは完全に空で、必要なときにシステムで使用できます。
- 非アクティブなメモリ— 「最近は」アクセスされていないページです。
- 合計メモリ — 実際の物理メモリ RAM のサイズです。
- 使用済みメモリ— プログラムやプロセスによって現在使用されているメモリです。

トピック

Linux: 収集されたメトリクスと使用された計算

収集されたメトリクスと単位:

- アクティブ (バイト)
- 使用可能 (バイト)
- 使用可能率 (パーセント)
- バッファ済み (バイト)
- キャッシュ済み (バイト)
- 空き (バイト)
- 非アクティブ (バイト)
- 合計 (バイト)
- 使用済み (バイト)
- 使用率 (パーセント)

使用済みメモリ = 合計メモリ - 空きメモリ - キャッシュされたメモリ - バッファメモリ

合計メモリ = 使用済みメモリ + 空きメモリ + キャッシュされたメモリ + バッファメモリ

macOS: 収集されたメトリクスと使用された計算

収集されたメトリクスと単位:

- アクティブ (バイト)
- 使用可能 (バイト)
- 使用可能率 (パーセント)
- 空き (バイト)
- 非アクティブ (バイト)
- 合計 (バイト)
- 使用済み (バイト)
- 使用率 (パーセント)

使用可能なメモリ = 空きメモリ + 非アクティブなメモリ

使用済みメモリ = 合計メモリ - 使用可能なメモリ

合計メモリ = 使用可能なメモリ + 使用済みメモリ

Windows: 収集されたメトリクス

Windows ホストで収集されたメトリクスは以下のとおりです。これらのメトリクスの `Unit` はすべて `None` です。

- 使用可能バイト
- キャッシュ障害数/秒
- ページ障害数/秒
- ページ数/秒

CloudWatch エージェントはパフォーマンスカウンターからイベントを解析するため、Windows メトリクスの計算は行われません。

例: Linux でのメモリメトリクスの計算

例として、Linux ホストで **`cat /proc/meminfo`** のコマンドを入力すると、次の結果が表示されるとします。

```
MemTotal:       3824388 kB
MemFree:         462704 kB
MemAvailable:   2157328 kB
Buffers:         126268 kB
Cached:         1560520 kB
SReclaimable:    289080 kB>
```

この例では、CloudWatch エージェントは次の値を収集します。CloudWatch エージェントが収集してレポートする値はすべてバイト単位です。

- `mem_total`: 3916173312 バイト
- `mem_available`: 2209103872 バイト (メモリフリー + キャッシュ)
- `mem_free`: 473808896 バイト
- `mem_cached`: 1893990400 バイト (`cached` + `SReclaimable`)
- `mem_used`: 1419075584 バイト (`MemTotal` — (`MemFree` + `Buffers` + (`Cached` + `SReclaimable`)))
- `mem_buffered`: 129667072 バイト
- `mem_available_percent`: 56.41%
- `mem_used_percent`: 36.24% (`mem_used` / `mem_total`) * 100

CloudWatch エージェントの一般的なシナリオ

以下のセクションでは、CloudWatch エージェントの一般的な設定と、カスタマイズタスクを完了する方法についての概要を説明します。

トピック

- CloudWatch エージェントの別のユーザーとしての実行
- CloudWatch エージェントがスパースログファイルを処理する方法
- CloudWatch エージェントにより収集されるメトリクスへのカスタムディメンションの追加
- CloudWatch エージェント設定ファイル
- CloudWatch エージェントによって収集されるメトリクスの集約またはロールアップ
- CloudWatch エージェントを使用した高解像度メトリクスの収集
- 別のアカウントへのメトリクス、ログ、トレースの送信

- 統合 CloudWatch エージェントと前の CloudWatch Logs エージェントの間のタイムスタンプの違い

CloudWatch エージェントの別のユーザーとしての実行

Linux サーバーでは、CloudWatch はデフォルトで root ユーザーとして実行されます。エージェントを別のユーザーとして実行するには、CloudWatch エージェント設定ファイルの `agent` セクションで `run_as_user` パラメータを使用します。このオプションは、Linux サーバーでのみ使用できます。

エージェントを root ユーザーとして既に実行している場合、別のユーザーに変更するには、以下のいずれかの手順を使用します。

Linux を実行する EC2 インスタンスで CloudWatch エージェントを別のユーザーとして実行するには

1. 新しい CloudWatch エージェントパッケージをダウンロードしてインストールします。詳細については、「CloudWatch エージェントパッケージをダウンロードする」を参照してください。
2. 新しい Linux ユーザーを作成するか、RPM ファイルまたは DEB ファイルで作成した `cwagent` という名前のデフォルトユーザーを使用します。
3. 次のいずれかの方法で、このユーザーの認証情報を指定します。
   - このファイル `.aws/credentials` が root ユーザーのホームディレクトリに存在する場合は、CloudWatch エージェントの実行に使用するユーザーの認証情報ファイルを作成する必要があります。この認証情報ファイルは `/home/username/.aws/credentials` になります。次に、`shared_credential_file` の `common-config.toml` パラメータの値を認証情報ファイルのパス名に設定します。詳細については、「(オプション) プロキシ情報またはリージョン情報の一般的な設定を変更する」を参照してください。
   - ファイル `.aws/credentials` が root ユーザーのホームディレクトリに存在しない場合は、次のいずれかの操作を実行できます。
     - CloudWatch エージェントの実行に使用するユーザーの認証情報ファイルを作成します。この認証情報ファイルは `/home/username/.aws/credentials` になります。次に、`shared_credential_file` の `common-config.toml` パラメータの値を認証情報ファイルのパス名に設定します。詳細については、「(オプション) プロキシ情報またはリージョン情報の一般的な設定を変更する」を参照してください。
     - 認証情報ファイルを作成する代わりに、IAM ロールをインスタンスにアタッチします。エージェントは、このロールを認証情報プロバイダーとして使用します。

4. CloudWatch エージェント設定ファイルで、agent セクションに次の行を追加します。

```
"run_as_user": "username"
```

必要に応じて、設定ファイルに他の変更を行います。詳細については、「CloudWatch エージェント設定ファイルを作成する」を参照してください。

5. 必要なアクセス許可をユーザーに付与します。ユーザーには、収集するログファイルの読み取り (r) アクセス許可と、ログファイルのパス内のすべてのディレクトリに対する実行 (x) アクセス許可が必要です。
6. 先ほど作成した設定ファイルを使用してエージェントを起動します。

```
sudo /opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-ctl -a fetch-config -m ec2 -s -c file:configuration-file-path
```

Linux を実行しているオンプレミスサーバーで CloudWatch エージェントを別のユーザーとして実行するには

1. 新しい CloudWatch エージェントパッケージをダウンロードしてインストールします。詳細については、「CloudWatch エージェントパッケージをダウンロードする」を参照してください。
2. 新しい Linux ユーザーを作成するか、RPM ファイルまたは DEB ファイルで作成した cwagent という名前のデフォルトユーザーを使用します。
3. このユーザーの認証情報をユーザーがアクセスできるパス (/home/*username*/.aws/credentials など) に保存します。
4. shared_credential_file の common-config.toml パラメータの値を認証情報ファイルのパス名に指定します。詳細については、「(オプション) プロキシ情報またはリージョン情報の一般的な設定を変更する」を参照してください。
5. CloudWatch エージェント設定ファイルで、agent セクションに次の行を追加します。

```
"run_as_user": "username"
```

必要に応じて、設定ファイルに他の変更を行います。詳細については、「CloudWatch エージェント設定ファイルを作成する」を参照してください。

6. 必要なアクセス許可をユーザーに付与します。ユーザーには、収集するログファイルの読み取り (r) アクセス許可と、ログファイルのパス内のすべてのディレクトリに対する実行 (x) アクセス許可が必要です。

7. 先ほど作成した設定ファイルを使用してエージェントを起動します。

```
sudo /opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-ctl -a fetch-config -m ec2 -s -c file:configuration-file-path
```

CloudWatch エージェントがスパースログファイルを処理する方法

スパースファイルは、空のブロックと実際のコンテンツの両方を持つファイルです。スパースファイルは、ブロックを構成する実際の null バイトの代わりに、空のブロックを表す簡単な情報をディスクに書き込むことによって、より効率的にディスク領域を使用します。これにより、スパースファイルの実際のサイズは、通常、見かけのサイズよりもずっと小さくなります。

ただし、CloudWatch エージェントはスパースファイルを通常のファイルと同じ方法で処理します。エージェントがスパースファイルを読み取ると、空のブロックは null バイトで満たされた「実」ブロックとして扱われます。このため、CloudWatch エージェントは、スパースファイルの見かけ上のサイズと同じバイト数を CloudWatch に発行します。

スパースファイルを発行するように CloudWatch エージェントを設定すると、予想よりも CloudWatch コストが高くなる可能性があるため、この設定を避けるようお勧めします。例えば、Linux の `/var/logs/lastlog` は非常にスパースなファイルであることが多いため、これを CloudWatch に発行しないようお勧めします。

CloudWatch エージェントにより収集されるメトリクスへのカスタムディメンションの追加

エージェントによって収集されるメトリクスにタグなどのカスタムディメンションを追加するには、それらのメトリクスをリストするエージェント設定ファイルのセクションに `append_dimensions` フィールドを追加します。

たとえば、設定ファイルの次のサンプルセクションでは、値が `stackName` の `Prod` というカスタムディメンションを、エージェントによって収集される `cpu` および `disk` メトリクスに追加します。

```
"cpu":{
  "resources":[
    "*"
  ],
  "measurement":[
    "cpu_usage_guest",
```

```
      "cpu_usage_nice",
      "cpu_usage_idle"
    ],
    "totalcpu":false,
    "append_dimensions":{
      "stackName":"Prod"
    }
  },
  "disk":{
    "resources":[
      "/",
      "/tmp"
    ],
    "measurement":[
      "total",
      "used"
    ],
    "append_dimensions":{
      "stackName":"Prod"
    }
  }
```

エージェント設定ファイルを変更するときは必ず、エージェントを再起動して変更を有効にする必要がある点に注意してください。

CloudWatch エージェント設定ファイル

Linux サーバーと Windows サーバーの両方で、複数の設定ファイルを使用するように CloudWatch エージェントを設定できます。例えば、インフラストラクチャ内のすべてのサーバーから常に収集する一連のメトリクス、ログ、およびトレースを収集する共通の設定ファイルを使用できます。その後、追加の設定ファイルを使用して、特定のアプリケーションから、または特定の状況でメトリクスを収集することができます。

これを設定するには、まず使用する設定ファイルを作成します。同じサーバー上で一緒に使用される設定ファイルは、異なるファイル名を持つ必要があります。設定ファイルは、サーバー上または Parameter Store に保存できます。

`fetch-config` オプションを使って CloudWatch エージェントを開始し、1 つ目の設定ファイルを指定します。エージェントの実行の 2 つ目の設定ファイルを追加するには、同じコマンドを使用しますが、`append-config` オプションを使います。いずれかの設定ファイルにリストされている、すべてのメトリクス、ログ、トレースが収集されます。次のコマンドは、このシナリオで設定スト

アをファイルとして使用した例を示しています。1 行目で infrastructure.json 設定ファイルを使ってエージェントを開始し、2 行目で app.json 設定ファイルを追加しています。

次のサンプルコマンドは、Linux 用です。

```
/opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-ctl -a fetch-config -m ec2
 -s -c file:/tmp/infrastructure.json
```

```
/opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-ctl -a append-config -m
 ec2 -s -c file:/tmp/app.json
```

次のコマンドは Windows Server 用です。

```
& "C:\Program Files\Amazon\AmazonCloudWatchAgent\amazon-cloudwatch-agent-ctl.ps1"
 -a fetch-config -m ec2 -s -c file:"C:\Program Files\Amazon\AmazonCloudWatchAgent
\infrastructure.json"
```

```
& "C:\Program Files\Amazon\AmazonCloudWatchAgent\amazon-cloudwatch-agent-ctl.ps1"
 -a append-config -m ec2 -s -c file:"C:\Program Files\Amazon\AmazonCloudWatchAgent
\app.json"
```

次の設定ファイルは、この機能の使用方法の例を示しています。1 つ目の設定ファイルは、インフラストラクチャ内のすべてのサーバーに使用され、2 つ目の設定ファイルは、特定のアプリケーションからのログのみを収集し、そのアプリケーションを実行しているサーバーに追加されています。

infrastructure.json

```
{
  "metrics": {
    "metrics_collected": {
      "cpu": {
        "resources": [
          "*"
        ],
        "measurement": [
          "usage_active"
        ],
        "totalcpu": true
      },
      "mem": {
         "measurement": [
```

```
            "used_percent"
        ]
      }
    }
  },
  "logs": {
    "logs_collected": {
      "files": {
        "collect_list": [
          {
            "file_path": "/opt/aws/amazon-cloudwatch-agent/logs/amazon-cloudwatch-
agent.log",
            "log_group_name": "amazon-cloudwatch-agent.log"
          },
          {
            "file_path": "/var/log/messages",
            "log_group_name": "/var/log/messages"
          }
        ]
      }
    }
  }
}
```

app.json

```
{
    "logs": {
        "logs_collected": {
            "files": {
                "collect_list": [
                    {
                        "file_path": "/app/app.log*",
                        "log_group_name": "/app/app.log"
                    }
                ]
            }
        }
    }
}
```

設定に追加される設定ファイルは、互いに異なるファイル名を持つ必要があり、また初期の設定ファイルと異なるファイル名を持つ必要があります。エージェントで既に使用されている設定ファイルと

同じファイル名の設定ファイルで append-config を使用した場合、その append コマンドでは追加は行われず、1 つ目の設定ファイルの情報は上書きされます。これは、同じファイル名の 2 つの設定ファイルが異なるファイルパスにある場合でも同様です。

前の例では、2 つの設定ファイルを使用していますが、エージェント設定に追加できる設定ファイルの数に制限はありません。サーバー上にある設定ファイルと Parameter Store にある設定を混在させることもできます。

CloudWatch エージェントによって収集されるメトリクスの集約またはロールアップ

エージェントにより収集されたメトリクスを集約 (ロールアップ) するには、aggregation_dimensions フィールドをエージェント設定ファイル内のそのメトリクスのセクションに追加します。

たとえば、次の設定ファイルスニペットでは、AutoScalingGroupName ディメンションでメトリクスをロールアップします。各 Auto Scaling グループ内のすべてのインスタンスのメトリクスが集約され、全体として参照できるようになります。

```
"metrics": {
  "cpu":{...}
  "disk":{...}
  "aggregation_dimensions" : [["AutoScalingGroupName"]]
}
```

Auto Scaling グループ名でのロールアップのほかに、各 InstanceId および InstanceType ディメンションの組み合わせでもロールアップするには、次の内容を追加します。

```
"metrics": {
  "cpu":{...}
  "disk":{...}
  "aggregation_dimensions" : [["AutoScalingGroupName"], ["InstanceId", "InstanceType"]]
}
```

代わりにメトリクスを 1 つのコレクションにロールアップするには、[] を使用します。

```
"metrics": {
  "cpu":{...}
  "disk":{...}
```

```
  "aggregation_dimensions" : [[]]
}
```

エージェント設定ファイルを変更するときは必ず、エージェントを再起動して変更を有効にする必要がある点に注意してください。

CloudWatch エージェントを使用した高解像度メトリクスの収集

metrics_collection_interval フィールドは、収集されるメトリクスの時間間隔 (秒単位) を指定します。このフィールドに 60 未満の値を指定することによって、メトリクスは、高解像度メトリクスとして収集されます。

たとえば、すべてのメトリクスを高解像度で 10 秒ごとに収集する必要がある場合は、metrics_collection_interval セクションで、グローバルメトリクス収集間隔として agent の値を 10 に指定します。

```
"agent": {
  "metrics_collection_interval": 10
}
```

または、次の例では cpu メトリクスを 1 秒ごとに収集し、他のすべてのメトリクスを 1 分ごとに収集するように設定します。

```
"agent":{
  "metrics_collection_interval": 60
},
"metrics":{
  "metrics_collected":{
    "cpu":{
      "resources":[
        "*"
      ],
      "measurement":[
        "cpu_usage_guest"
      ],
      "totalcpu":false,
      "metrics_collection_interval": 1
    },
    "disk":{
      "resources":[
        "/",
```

```
        "/tmp"
      ],
      "measurement":[
        "total",
        "used"
      ]
    }
  }
}
```

エージェント設定ファイルを変更するときは必ず、エージェントを再起動して変更を有効にする必要がある点に注意してください。

別のアカウントへのメトリクス、ログ、トレースの送信

CloudWatch エージェントがメトリクス、ログ、またはトレースを別のアカウントに送信するには、送信サーバーのエージェント設定ファイルで role_arn パラメータを指定します。role_arn 値は、エージェントがターゲットアカウントにデータを送信する際に使用するターゲットアカウントの IAM ロールを指定します。このロールにより、メトリクスまたはログをターゲットアカウントに配信するときに、送信アカウントがターゲットアカウントの対応するロールを担うことができます。

エージェント設定ファイルには、個別の role_arn 文字列を指定することもできます。1 つはメトリクスを送信するときに、1 つはログを送信するために、もう 1 つはトレースを送信するために使用します。

次の設定ファイルの agent セクションの一部の例は、データを別のアカウントに送信するときにエージェントが CrossAccountAgentRole を使用するように設定します。

```
{
  "agent": {
    "credentials": {
      "role_arn": "arn:aws:iam::123456789012:role/CrossAccountAgentRole"
    }
  },
  .....
}
```

または、次の例では、送信アカウントがメトリクス、ログ、トレースを送信するために使用する別々のロールを設定しています。

```
"metrics": {
    "credentials": {
     "role_arn": "RoleToSendMetrics"
    },
    "metrics_collected": {....
```

```
"logs": {
    "credentials": {
    "role_arn": "RoleToSendLogs"
    },
    ....
```

必要なポリシー

エージェント設定ファイルで role_arn を指定する場合は、送信アカウントとターゲットアカウントの IAM ロールに特定のポリシーが設定されていることも確認する必要があります。送信アカウントとターゲットアカウントの両方のロールに、[CloudWatchAgentServerPolicy] が必要です。このポリシーをロールに割り当てる方法の詳細については、「Amazon EC2 インスタンスの CloudWatch エージェントで使用する IAM ロールを作成する」を参照してください。

また、送信側アカウントのロールに次のポリシーも含まれている必要があります。このポリシーは、ロールを編集するときに IAM コンソールの [アクセス許可] タブで追加します。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "sts:AssumeRole"
            ],
            "Resource": [
                "arn:aws:iam::target-account-ID:role/agent-role-in-target-account"
            ]
        }
    ]
}
```

また、ターゲットアカウントのロールには、送信側アカウントで使用される IAM ロールを認識できるように、次のポリシーが含まれている必要があります。このポリシーは、ロールを編集するとき

に IAM コンソールの [信頼関係] タブで追加します。このポリシーを追加するターゲットアカウントのロールは、CloudWatch エージェントで使用する IAM ロールとユーザーを作成する で作成したロールです。このロールは、送信側アカウントで使用されるポリシーの *agent-role-in-target-account* で指定したロールです。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": [
          "arn:aws:iam::sending-account-ID:role/role-in-sender-account"
        ]
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

統合 CloudWatch エージェントと前の CloudWatch Logs エージェントの間のタイムスタンプの違い

CloudWatch エージェントでは、前の CloudWatch Logs エージェントと比べて、タイムスタンプ形式のさまざまな記号のセットがサポートされています。これらの違いを次の表に示します。

両方のエージェントがサポートする記号	統合 CloudWatch エージェントのみがサポートする記号	前の CloudWatch Logs エージェントでのみサポートされている記号
%A、%a、%b、%B、%d、%f、%H、%I、%m、%M、%p、%S、%y、%Y、%Z、%z	%-d、%-I、%m、%M、%-S	%c、% j、%U、%W、%w

新しい CloudWatch エージェントでサポートされる記号の意味の詳細については、Amazon CloudWatch ユーザーガイドの「CloudWatch エージェント設定ファイル: ログセクション」を

参照してください。CloudWatch Logs エージェントでサポートされる記号については、Amazon CloudWatch Logs ユーザーガイドの「エージェント設定ファイル」を参照してください。

CloudWatch エージェントのトラブルシューティング

以下の情報を参考にして、CloudWatch エージェントに関する問題のトラブルシューティングを行います。

トピック

CloudWatch エージェントのコマンドラインパラメータ

CloudWatch エージェントでサポートされているパラメータの完全なリストを表示するには、エージェントがインストールされているコンピュータで、コマンドラインに以下のコマンドを入力します。

```
amazon-cloudwatch-agent-ctl -help
```

Run Command を使用した CloudWatch エージェントのインストールが失敗する

Systems Manager Run Command を使用して CloudWatch エージェントをインストールするには、ターゲットサーバー上の SSM Agent がバージョン 2.2.93.0 以降である必要があります。SSM Agent のバージョンが適切でない場合は、次のエラーメッセージが表示されることがあります。

```
no latest version found for package AmazonCloudWatchAgent on platform linux
```

```
failed to download installation package reliably
```

SSM Agent バージョンの更新については、AWS Systems Manager ユーザーガイドの SSM Agent のインストールと設定を参照してください。

CloudWatch エージェントが開始されない

CloudWatch エージェントが開始されない場合、設定に問題がある可能性があります。設定情報は、`configuration-validation.log` ファイルに口グ記録されます。このファイルは、Linux サーバーでは `/opt/aws/amazon-cloudwatch-agent/logs/configuration-validation.log` に、Windows Server を実行しているサーバーでは `$Env:ProgramData\Amazon\AmazonCloudWatchAgent\Logs\configuration-validation.log` にあります。

CloudWatch エージェントが実行されていることを確認する

CloudWatch エージェントのクエリを実行して、実行中か停止中かを調べることができます。これをリモートに行うには、AWS Systems Manager を使用できます。コマンドラインを使用することもできますが、チェックできるのはローカルサーバーだけです。

Run Command を使用して CloudWatch エージェントのステータスをクエリするには

1. Systems Manager コンソール (https://console.aws.amazon.com/systems-manager/) を開きます。
2. ナビゲーションペインで、[Run Command] を選択します。

   -または-

   AWS Systems Manager のホームページが表示された場合は、スクロールダウンして [Explore Run Command] (Run Command の詳細) を選択します。

3. [Run command (コマンドの実行)] を選択します。
4. [コマンドドキュメント] リストで、[AmazonCloudWatch-ManageAgent] の横のボタンをクリックします。
5. [Action] リストで、[status] を選択します。
6. [オプションの設定ソース] で [デフォルト] を選択し、[オプションの設定場所] を空白のままにします。
7. [Target] 領域で、確認するインスタンスを選択します。
8. [Run (実行)] を選択します。

エージェントが実行されている場合、出力は次のようになります。

```
{
       "status": "running",
       "starttime": "2017-12-12T18:41:18",
       "version": "1.73.4"
}
```

エージェントが停止した場合、"status" フィールドには "stopped" が表示されます。

コマンドラインを使用して CloudWatch エージェントのステータスのクエリをローカルで実行するには

- Linux サーバーで、次のように入力します。

  ```
  sudo /opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-ctl -m ec2 -a
   status
  ```

  Windows Server を実行しているサーバーの場合、管理者として PowerShell で次のように入力します。

  ```
  & $Env:ProgramFiles\Amazon\AmazonCloudWatchAgent\amazon-cloudwatch-agent-ctl.ps1 -m
   ec2 -a status
  ```

CloudWatch エージェントが起動せず、Amazon EC2 リージョンに関するエラーが発生する

エージェントが起動せず、エラーメッセージに Amazon EC2 リージョンのエンドポイントが示されている場合は、Amazon EC2 エンドポイントへのアクセス権を必要とするエージェントを設定しただけで、そのアクセス権を付与していない可能性があります。

例えば、エージェントの設定ファイルで Amazon EC2 メタデータに依存する値を `append_dimensions` パラメータに指定し、プロキシー使用する場合は、そのサーバーが Amazon EC2 のエンドポイントにアクセスできることを確認する必要があります。これらのエンドポイントの詳細については、「Amazon Web Services 全般のリファレンス」の「Amazon Elastic Compute Cloud (Amazon EC2)」を参照してください。

CloudWatch エージェントが Windows Server で開始されない

Windows Server では、次のエラーが表示されることがあります。

```
Start-Service : Service 'Amazon CloudWatch Agent (AmazonCloudWatchAgent)' cannot be
 started due to the following
error: Cannot start service AmazonCloudWatchAgent on computer '.'.
At C:\Program Files\Amazon\AmazonCloudWatchAgent\amazon-cloudwatch-agent-ctl.ps1:113
 char:12
+     $svc | Start-Service
+            ~~~~~~~~~~~~~
    + CategoryInfo          : OpenError:
 (System.ServiceProcess.ServiceController:ServiceController) [Start-Service],
   ServiceCommandException
    + FullyQualifiedErrorId :
 CouldNotStartService,Microsoft.PowerShell.Commands.StartServiceCommand
```

これを修正するには、まずサーバーサービスが実行中であることを確認します。このエラーは、サーバーサービスが実行されていないときにエージェントが起動しようとした場合に表示されます。

サーバーサービスがすでに実行されている場合は、以下の点が問題である可能性があります。一部の Windows Server インストールでは、CloudWatch エージェントの起動に 30 秒以上かかります。Windows Server では、サービスの起動に許容されるデフォルトの時間が 30 秒のみであるため、エージェントが失敗して次のようなエラーが表示される場合があります。

この問題を修正するには、サービスのタイムアウト値を増やします。詳細については、「サービスが開始されず、イベント 7000 および 7011 が Windows イベントログに記録される」を参照してください。

メトリクスの場所

CloudWatch エージェントが実行されているが、そのエージェントによって収集されたメトリクスが AWS Management Console または AWS CLI で見つからない場合、適切な名前空間を使用していることを確認します。エージェントによって収集されるメトリクスの名前空間は、デフォルトでは CWAgent です。この名前空間は、エージェント設定ファイルの [namespace] セクションの [metrics] フィールドを使用してカスタマイズできます。予想されるメトリクスが表示されない場合、設定ファイルで、使用している名前空間を確認してください。

CloudWatch エージェントパッケージを初めてダウンロードした場合、エージェント設定ファイルは amazon-cloudwatch-agent.json です。このファイルは、設定ウィザードを実行したディレクトリにあります。または、別のディレクトリに自分で移動した可能性があります。設定ウィザードを使用した場合は、ウィザードからのエージェント設定ファイル出力の名前は config.json です。namespace フィールドなど、設定ファイルの詳細については、「 CloudWatch エージェント設定ファイル: Metrics セクション」を参照してください。

CloudWatch エージェントが、コンテナで実行に長い時間がかかる、またはホップ制限のエラーをログに記録する

CloudWatch エージェントをコンテナサービスとして実行し、エージェントによって収集されたすべてのメトリクスに Amazon EC2 メトリクスのディメンションを追加する場合、エージェントのバージョン v1.247354.0 で次のエラーが表示されることがあります。

```
2022-06-07T03:36:11Z E! [processors.ec2tagger] ec2tagger: Unable to retrieve Instance
 Metadata Tags. This plugin must only be used on an EC2 instance.
2022-06-07T03:36:11Z E! [processors.ec2tagger] ec2tagger: Please increase hop limit
 to 2 by following this document https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/
configuring-instance-metadata-options.html#configuring-IMDS-existing-instances.
2022-06-07T03:36:11Z E! [telegraf] Error running agent: could not initialize processor
 ec2tagger: EC2MetadataRequestError: failed to get EC2 instance identity document
caused by: EC2MetadataError: failed to make EC2Metadata request
        status code: 401, request id:
caused by:
```

このエラーは、エージェントが適切なホップ制限なしでコンテナ内の IMDSv2 からメタデータを取得しようとした場合に表示されることがあります。v1.247354.0 より前のバージョンのエージェントでは、ログメッセージが表示されることなくこの問題が発生する可能性があります。

これを解決するには、「インスタンスメタデータオプションの設定」の指示に従い、ホップ制限を 2 に増やします。

エージェントの設定を更新しましたが、CloudWatch コンソールに新しいメトリクスやログが表示されません

CloudWatch エージェント設定ファイルを更新した場合、次回にエージェントを起動したときに、**fetch-config** オプションを使用する必要があります。例えば、更新したファイルをローカルコンピュータに保存した場合は、次のコマンドを入力します。

```
sudo /opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-ctl -a fetch-config -s -m ec2 -c file:configuration-file-path
```

CloudWatch エージェントファイルとロケーション

次の表は、Linux や Windows Server を実行しているサーバー上に CloudWatch エージェントによってインストールされて使用されるファイルおよびインストール先の一覧です。

ファイル	Linux のロケーション	Windows Server のロケーション
エージェントの起動、停止、再起動を制御する制御スクリプト。	/opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-ctl 、、または /usr/bin/amazon-cloudwatch-agent-ctl	$Env:ProgramFiles\Amazon\AmazonCloudWatchAgent\amazon-cloudwatch-agent-ctl.ps1
エージェントが書き込むログファイル。AWS Support に問い合わせる際は、このファイルを添付する必要があります。	/opt/aws/amazon-cloudwatch-agent/logs/amazon-cloudwatch-agent.log 、、または /var/log/amazon/amazon-cloudwatch-ag	$Env:ProgramData\Amazon\AmazonCloudWatchAgent\Logs\amazon-cloudwatch-agent.log

ファイル	Linux のロケーション	Windows Server のロケーション
	ent/amazon-cloudwatch-agent.log	
エージェント妥当性確認検証ファイル。	/opt/aws/amazon-cloudwatch-agent/logs/configuration-validation.log 、、または /var/log/amazon/amazon-cloudwatch-agent/configuration-validation.log	$Env:ProgramData\Amazon\AmazonCloudWatchAgent\Logs\configuration-validation.log
ウィザードが作成した直後に、エージェントの設定に使用される JSON ファイル。詳細については、「CloudWatch エージェント設定ファイルを作成する」を参照してください。	/opt/aws/amazon-cloudwatch-agent/bin/config.json	$Env:ProgramFiles\Amazon\AmazonCloudWatchAgent\config.json
この設定ファイルが Parameter Store からダウンロードした場合に、エージェントの設定に使用される JSON ファイル。	/opt/aws/amazon-cloudwatch-agent/etc/amazon-cloudwatch-agent.json 、、または /etc/amazon/amazon-cloudwatch-agent/amazon-cloudwatch-agent.json	$Env:ProgramData\Amazon\AmazonCloudWatchAgent\amazon-cloudwatch-agent.json

ファイル	Linux のロケーション	Windows Server のロケーション
エージェントで使用するリージョン情報と認証情報を指定するための TOML ファイル (システムのデフォルト値を上書きします)。	/opt/aws/amazon-cloudwatch-agent/etc/common-config.toml 、または /etc/amazon/amazon-cloudwatch-agent/common-config.toml	$Env:ProgramData\Amazon\AmazonCloudWatchAgent\common-config.toml
JSON 設定ファイルの変換された内容を含む TOML ファイル。amazon-cloudwatch-agent-ctl スクリプトは、このファイルを生成します。このファイルは直接変更しないでください。JSON から TOML への変換が正常に完了したことを確認するのに役立ちます。	/opt/aws/amazon-cloudwatch-agent/etc/amazon-cloudwatch-agent.toml 、または /etc/amazon/amazon-cloudwatch-agent/amazon-cloudwatch-agent.toml	$Env:ProgramData\Amazon\AmazonCloudWatchAgent\amazon-cloudwatch-agent.toml
JSON 設定ファイルの変換された内容を含む YAML ファイル。amazon-cloudwatch-agent-ctl スクリプトは、このファイルを生成します。このファイルは直接変更しないでください。このファイルは、JSON から YAML への変換が正常に完了したことを確認するのに役立ちます。	/opt/aws/amazon-cloudwatch-agent/etc/amazon-cloudwatch-agent.yaml or /etc/amazon/amazon-cloudwatch-agent/amazon-cloudwatch-agent.yaml	$Env:ProgramData\Amazon\AmazonCloudWatchAgent\amazon-cloudwatch-agent.yaml

CloudWatch エージェントのバージョンについての情報の検索

Linux サーバー上の CloudWatch エージェントのバージョン番号を検索するには、次のコマンドを入力します。

```
sudo /opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-ctl -a status
```

Windows Server 上の CloudWatch エージェントのバージョン番号を検索するには、次のコマンドを入力します。

```
& $Env:ProgramFiles\Amazon\AmazonCloudWatchAgent\amazon-cloudwatch-agent-ctl.ps1 -m ec2
 -a status
```

> **Note**
>
> このコマンドを使用することは、CloudWatch エージェントのバージョンを検索する正しい方法です。コントロールパネルの [Programs and Features (プログラムと機能)] を使用すると、誤ったバージョン番号が表示されます。

エージェントに対する最新の変更に関する README ファイル、およびダウンロード可能なバージョン番号を示すファイルをダウンロードすることもできます。これらのファイルは次の場所にあります。

- `https://amazoncloudwatch-agent.s3.amazonaws.com/info/latest/RELEASE_NOTES` 、、または `https://amazoncloudwatch-agent-`*`region`*`.s3.`*`region`*`.amazonaws.com/info/latest/RELEASE_NOTES`
- `https://amazoncloudwatch-agent.s3.amazonaws.com/info/latest/CWAGENT_VERSION` 、、または `https://amazoncloudwatch-agent-`*`region`*`.s3.`*`region`*`.amazonaws.com/info/latest/CWAGENT_VERSION`

CloudWatch エージェントによって生成されたログ

エージェントは、実行されている間ログを生成します。このログには、トラブルシューティング情報が含まれています。このログは、amazon-cloudwatch-agent.log ファイルで確認できます。このファイルは、Linux サーバーでは /opt/aws/amazon-cloudwatch-agent/logs/amazon-

cloudwatch-agent.log に、Windows Server を実行しているサーバーでは $Env:ProgramData\Amazon\AmazonCloudWatchAgent\Logs\amazon-cloudwatch-agent.log にあります。

amazon-cloudwatch-agent.log ファイルに追加の詳細を記録するようにエージェントを設定できます。エージェント設定ファイルの agent セクションで、debug フィールドを true に設定した後、CloudWatch エージェントを再設定して再起動します。この追加情報のログ記録を無効にするには、debug フィールドを false に設定します。次に、エージェントを再設定して再起動します。(詳細については、 CloudWatch エージェント設定ファイルを手動で作成または編集する を参照してください)。

バージョン 1.247350.0 以降の CloudWatch エージェントでは、必要に応じて、エージェント設定ファイルの agent セクションの aws_sdk_log_level フィールドに、次のオプションの 1 つまたは複数を指定できます。複数のオプションは、| 文字で区切ります。

- LogDebug
- LogDebugWithSigning
- LogDebugWithHTTPBody
- LogDebugRequestRetries
- LogDebugWithEventStreamBody

これらのオプションの詳細については、「LogLevelType」を参照してください。

CloudWatch エージェントの停止と再起動

AWS Systems Manager またはコマンドラインを使用して CloudWatch エージェントを手動で停止できます。

Run Command を使用して CloudWatch エージェントを停止するには

1. Systems Manager コンソール (https://console.aws.amazon.com/systems-manager/) を開きます。
2. ナビゲーションペインで、[Run Command] を選択します。

   -または-

   AWS Systems Manager のホームページが表示された場合は、スクロールダウンして [Explore Run Command] (Run Command の詳細) を選択します。
3. [Run command (コマンドの実行)] を選択します。

4. [Command document] リストで、[AmazonCloudWatch-ManageAgent] を選択します。
5. [Targets] 領域で、CloudWatch エージェントをインストールしたインスタンスを選択します。
6. [Action] リストで、[stop] を選択します。
7. [Optional Configuration Source] と [Optional Configuration Location] は空白のままにします。
8. [Run (実行)] を選択します。

コマンドラインを使用して CloudWatch エージェントをローカルで停止するには

- Linux サーバーで、次のように入力します。

```
sudo /opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-ctl -m ec2 -a
 stop
```

  Windows Server を実行しているサーバーの場合、管理者として PowerShell で次のように入力します。

```
& $Env:ProgramFiles\Amazon\AmazonCloudWatchAgent\amazon-cloudwatch-agent-ctl.ps1 -m
 ec2 -a stop
```

エージェントを再起動するには、「CloudWatch エージェントを起動する」の手順に従います。

ログ内へのメトリクスの埋め込み

CloudWatch 埋め込みメトリクス形式を使用すると、CloudWatch Logs に書き込まれるログの形式でカスタムメトリクスを非同期的に生成できます。詳細なログイベントデータと一緒にカスタムメトリクスを埋め込むことができます。CloudWatch はカスタムメトリクスを自動的に抽出し、リアルタイムのインシデント検出のためにそれらを可視化して警告できるようにします。また、CloudWatch Logs Insights を使用して、抽出されたメトリクスに関連付けられた詳細なログイベントを照会し、運用イベントの根本原因に関する深い洞察を提供できます。

埋め込みメトリクスフォーマットは、Lambda 関数やコンテナなどの一時的なリソースから実用的なカスタムメトリクスを生成するのに役立ちます。埋め込みメトリックフォーマットを使用してこれらの一時的なリソースからログを送信することにより、ログデータの強力な分析機能を獲得しながら、個別のコードをインストルメント化または保守することなく、カスタムメトリックを簡単に作成できるようになりました。

埋め込みメトリクス形式を使用するために設定は必要ありません。埋め込みメトリクス形式の仕様に従ってログを構造化するか、クライアントライブラリを使用してログを生成し、PutLogEvents API または CloudWatch エージェントを使用して CloudWatch Logs に送信します。

ログの取り込みとアーカイブ、および生成されるカスタムメトリックには料金が発生します。詳細については、Amazon CloudWatch 料金表をご覧ください。

> **Note**
>
> カスタムメトリックの使用状況と対応する請求に影響するため、メトリックの抽出を設定するときには注意が必要です。高カーディナリティディメンション (`requestId` など) に基づいて意図せずにメトリックを作成すると、埋め込みメトリックフォーマットにより、各一意のディメンションの組み合わせに対応するカスタムメトリックが意図的に作成されます。詳細については、「ディメンション」を参照してください。

トピック

埋め込みメトリクス形式を使用したログの発行

次の方法で埋め込みメトリックフォーマットログを生成できます。

- オープンソースのクライアントライブラリを使用してログを生成して送信する
- 埋め込みメトリクス形式の仕様を使用してログを手動で生成し、CloudWatch エージェントまたは PutLogEvents API を使用してログを送信します。

トピック

クライアントライブラリを使用した埋め込みメトリクス形式でのログの作成

Amazon では、埋め込みメトリックフォーマットログの作成に使用できるオープンソースのクライアントライブラリを提供しています。現在、これらのライブラリは次の言語で使用できます。さまざまな設定の完全な例は、/examples の下のクライアントライブラリにあります。

ライブラリとその使用方法の手順は Github にあります。次のリンクを使用します。

- Node.Js

> **Note**
>
> Node.js の場合、Lambda JSON ログ形式を使用するには、バージョン 4.1.1 以降、3.0.2 以降、2.0.7 以降が必要です。このような Lambda 環境で以前のバージョンを使用すると、メトリクスが失われます。
> 詳細については、「AWS Lambda の Amazon CloudWatch Logs へのアクセス」を参照してください。

- Python

- Java
- C#

クライアントライブラリは、すぐに CloudWatch エージェントで動作するように設計されています。生成された埋め込みメトリクス形式のログは CloudWatch エージェントに送信され、CloudWatch エージェントはそれらを集約して CloudWatch Logs に発行します。

> **Note**
>
> Lambda を使用する場合、CloudWatch にログを送信するエージェントは必要ありません。STDOUT に口グ記録されたものはすべて、Lambda Logging Agent 経由で CloudWatch Logs に送信されます。

仕様: 埋め込みメトリクスフォーマット

CloudWatch 埋め込みメトリクスフォーマットは、構造化口グイベントに埋め込まれたメトリクス値を自動的に抽出するように CloudWatch Logs に指示するために使用される JSON 仕様です。CloudWatch を使用して、抽出されたメトリクス値のアラームをグラフ化および作成できます。

埋め込みメトリクスフォーマット仕様の表記規則

このフォーマット仕様では、「する必要がある」、「することはできない」、「必須」、「しなければならない」、「してはならない」、「すべきである」、「すべきではない」、「推奨」、「することができる」、「オプション」というキーワードは、キーワード RFC 2119で説明されているように解釈されます。

このフォーマット仕様では、「JSON」、「JSON テキスト」、「JSON 値」、「メンバー」、「要素」、「オブジェクト」、「配列」、「数値」、「文字列」、「boolean」、「true」、「false」、「null」という用語は、JavaScript Object Notation RFC8259 で定義されているように解釈されます。

> **Note**
>
> 埋め込みメトリクス形式を使用して作成されたメトリクスに対してアラームを作成する予定がある場合は、「埋め込みメトリクス形式で作成されたメトリクスにアラームを設定する」の推奨事項を参照してください。

埋め込みメトリクスフォーマットドキュメントの構造

このセクションでは、埋め込みメトリクスフォーマットドキュメントの構造について説明します。埋め込みメトリックフォーマットドキュメントは、JavaScript Object Notation RFC8259 で定義されています。

特に明記されていない限り、この仕様で定義されたオブジェクトに追加のメンバーを含めることはできません。この仕様で認識されないメンバーは無視する必要があります。この仕様で定義されているメンバーは、大文字と小文字が区別されます。

埋め込みメトリクスフォーマットは、標準 CloudWatch Logs イベントと同じ制限が適用され、最大サイズは 256 KB に制限されます。

埋め込みメトリクス形式を使用すると、アカウントの AWS/Logs 名前空間で公開されるメトリクスによって、EMF 口グの処理を追跡することができます。これらのメトリクスは、EMF からのメトリクス生成の失敗を追跡するために使用できます。また、障害が発生したのは解析によるものなのか、検証によるものなのかを追跡できます。詳細については、「CloudWatch メトリクスによるモニタリング」を参照してください。

ルートノード

LogEvent メッセージは、LogEvent メッセージ文字列の先頭または末尾に追加データがない有効な JSON オブジェクトである必要があります。LogEvent 構造の詳細については、「InputLogEvent」を参照してください。

埋め込みメトリックフォーマットドキュメントには、ルートノード上の次の最上位メンバーが含まれている必要があります。これは メタデータオブジェクト オブジェクトです。

```
{
 "_aws": {
    "CloudWatchMetrics": [ ... ]
  }
}
```

ルートノードには、MetricDirective オブジェクト の参照によって定義されたすべての ターゲットメンバー メンバーが含まれている必要があります。

ルートノードには、上記の要件に含まれていない他のメンバーを含めることができます。これらのメンバーの値は有効な JSON 型である必要があります。

メタデータオブジェクト

_aws メンバーを使用して、ダウンストリームサービスに LogEvent の処理方法を通知するペイロードに関するメタデータを表すことができます。値はオブジェクトでなければならず、次のメンバーが含まれている必要があります。

- CloudWatchMetrics – LogEvent のルートノードからメトリクスを抽出するように CloudWatch に指示するために使用される MetricDirective オブジェクト の配列。

```
{
  "_aws": {
    "CloudWatchMetrics": [ ... ]
  }
}
```

- Timestamp – イベントから抽出されたメトリクスに使用されるタイムスタンプを表す数値。値は、1970 年 1 月 1 日 00:00:00 UTC からのミリ秒数で表される必要があります。

```
{
  "_aws": {
    "Timestamp": 1559748430481
  }
}
```

MetricDirective オブジェクト

MetricDirective オブジェクトは、CloudWatch に抽出および発行されるメトリクスが LogEvent に含まれていることをダウンストリームサービスに通知します。MetricDirectives には、次のメンバーが含まれている必要があります。

- Namespace – メトリクスの CloudWatch 名前空間を表す文字列。
- Dimensions – DimensionSet 配列。
- Metrics – MetricDefinition オブジェクトの配列。この配列に、100 個を超える MetricDefinition オブジェクトを含めることはできません。

DimensionSet 配列

DimensionSet は、ドキュメント内のすべてのメトリックに適用されるディメンションキーを含む文字列の配列です。この配列内の値は、ターゲットメンバー と呼ばれる、ルートノード上のメンバーである必要もあります。

DimensionSet に、30 個を超えるディメンションキーを含めることはできません。DimensionSet が空である可能性があります。

ターゲットメンバーには文字列値が必要です。この値には、1024 文字以上を含めることはできません。ターゲットメンバーにより、メトリック ID の一部として発行されるディメンションが定義されます。使用する DimensionSet ごとに、CloudWatch に新しいメトリクスが作成されます。ディメンションの詳細については、「Dimension」と「ディメンション」を参照してください。

```
{
 "_aws": {
   "CloudWatchMetrics": [
     {
       "Dimensions": [ [ "functionVersion" ] ],
       ...
     }
   ]
 },
 "functionVersion": "$LATEST"
}
```

> **Note**
>
> カスタムメトリックの使用状況と対応する請求に影響するため、メトリックの抽出を設定するときには注意が必要です。高カーディナリティディメンション (`requestId` など) に基づいて意図せずにメトリックを作成すると、埋め込みメトリックフォーマットにより、各一意のディメンションの組み合わせに対応するカスタムメトリックが意図的に作成されます。詳細については、「ディメンション」を参照してください。

MetricDefinition オブジェクト

MetricDefinition は、次のメンバーを含める必要があるオブジェクトです。

- Name – メトリクス ターゲットメンバー に対する文字列 参照値。メトリックターゲットは、数値または数値の配列でなければなりません。

MetricDefinition オブジェクトには、以下のメンバーを含めることができます。

- Unit – 対応するメトリクスの測定単位を表すオプションの文字列値。値は有効な CloudWatch メトリクス単位である必要があります。有効な単位については、「MetricDatum」を参照してください。値を指定しない場合は、デフォルト値の NONE が使用されます。
- StorageResolution – 対応するメトリクスのストレージ解像度を表すオプションの整数値。これを 1 に設定すると、このメトリクスが高解像度メトリクスとして指定されるので、CloudWatch は 1 分未満 (最小 1 秒) の解像度でメトリクスを保存します。これを 60 に設定すると、このメトリクスが標準解像度メトリクスとして指定され、CloudWatch は 1 分の解像度でメトリクスを保存します。値は、CloudWatch がサポートする有効な解像度 (1 または 60) にする必要があります。値が指定されない場合は、デフォルト値の 60 が想定されます。

  高解像度メトリクスの詳細については、「高解像度のメトリクス」を参照してください。

> **Note**
>
> 埋め込みメトリクス形式を使用して作成されたメトリクスに対してアラームを作成する予定がある場合は、「埋め込みメトリクス形式で作成されたメトリクスにアラームを設定する」の推奨事項を参照してください。

```
{
  "_aws": {
    "CloudWatchMetrics": [
      {
        "Metrics": [
          {
            "Name": "Time",
            "Unit": "Milliseconds",
            "StorageResolution": 60
          }
        ],
        ...
      }
    ]
  },
  "Time": 1
}
```

参照値

参照値は、ルートノード上の ターゲットメンバー メンバーを参照する文字列値です。これらの参照は、RFC6901 で説明されている JSON ポインタと混同しないでください。ターゲット値をネストすることはできません。

ターゲットメンバー

有効なターゲットは、ルートノード上のメンバーでなければならず、ネストされたオブジェクトにすることはできません。たとえば、"A.a" の _reference_ 値は、次のメンバーと一致する必要があります。

```
{ "A.a" }
```

ネストされたメンバーと一致してはなりません。

```
{ "A": { "a" } }
```

ターゲットメンバーの有効な値は、それらを参照しているものによって異なります。メトリクスターゲットは、数値または数値の配列である必要があります。数値配列メトリクスターゲットは、100 を超えるメンバーを持つことはできません。ディメンションターゲットには文字列値が必要です。

埋め込みメトリクスフォーマットの例と JSON スキーマ

次に、埋め込みメトリックフォーマットの有効な例を示します。

```
{
  "_aws": {
    "Timestamp": 1574109732004,
    "CloudWatchMetrics": [
      {
        "Namespace": "lambda-function-metrics",
        "Dimensions": [["functionVersion"]],
        "Metrics": [
          {
            "Name": "time",
            "Unit": "Milliseconds",
            "StorageResolution": 60
          }
        ]
      }
```

```
    ]
  },
  "functionVersion": "$LATEST",
  "time": 100,
  "requestId": "989ffbf8-9ace-4817-a57c-e4dd734019ee"
}
```

次のスキーマを使用して、埋め込みメトリックフォーマットドキュメントを検証できます。

```
{
    "type": "object",
    "title": "Root Node",
    "required": [
        "_aws"
    ],
    "properties": {
        "_aws": {
            "$id": "#/properties/_aws",
            "type": "object",
            "title": "Metadata",
            "required": [
                "Timestamp",
                "CloudWatchMetrics"
            ],
            "properties": {
                "Timestamp": {
                    "$id": "#/properties/_aws/properties/Timestamp",
                    "type": "integer",
                    "title": "The Timestamp Schema",
                    "examples": [
                        1565375354953
                    ]
                },
                "CloudWatchMetrics": {
                    "$id": "#/properties/_aws/properties/CloudWatchMetrics",
                    "type": "array",
                    "title": "MetricDirectives",
                    "items": {
                        "$id": "#/properties/_aws/properties/CloudWatchMetrics/items",
                        "type": "object",
                        "title": "MetricDirective",
                        "required": [
                            "Namespace",
```

```
                            "Dimensions",
                            "Metrics"
                        ],
                        "properties": {
                            "Namespace": {
                                "$id": "#/properties/_aws/properties/CloudWatchMetrics/
items/properties/Namespace",
                                "type": "string",
                                "title": "CloudWatch Metrics Namespace",
                                "examples": [
                                    "MyApp"
                                ],
                                "pattern": "^(.*)$",
                                "minLength": 1,
                                "maxLength": 1024
                            },
                            "Dimensions": {
                                "$id": "#/properties/_aws/properties/CloudWatchMetrics/
items/properties/Dimensions",
                                "type": "array",
                                "title": "The Dimensions Schema",
                                "minItems": 1,
                                "items": {
                                    "$id": "#/properties/_aws/properties/
CloudWatchMetrics/items/properties/Dimensions/items",
                                    "type": "array",
                                    "title": "DimensionSet",
                                    "minItems": 0,
                                    "maxItems": 30,
                                    "items": {
                                        "$id": "#/properties/_aws/properties/
CloudWatchMetrics/items/properties/Dimensions/items/items",
                                        "type": "string",
                                        "title": "DimensionReference",
                                        "examples": [
                                            "Operation"
                                        ],
                                        "pattern": "^(.*)$",
                                        "minLength": 1,
                                        "maxLength": 250
}
                                    }
                            },
                            "Metrics": {
```

```
                                "$id": "#/properties/_aws/properties/CloudWatchMetrics/
items/properties/Metrics",
                                "type": "array",
                                "title": "MetricDefinitions",
                                "items": {
                                    "$id": "#/properties/_aws/properties/
CloudWatchMetrics/items/properties/Metrics/items",
                                    "type": "object",
                                    "title": "MetricDefinition",
                                    "required": [
                                        "Name"
                                    ],
                                    "properties": {
                                        "Name": {
                                            "$id": "#/properties/_aws/properties/
CloudWatchMetrics/items/properties/Metrics/items/properties/Name",
                                            "type": "string",
                                            "title": "MetricName",
                                            "examples": [
                                                "ProcessingLatency"
                                            ],
                                            "pattern": "^(.*)$",
                                            "minLength": 1,
                                            "maxLength": 1024
                                        },
                                        "Unit": {
                                            "$id": "#/properties/_aws/properties/
CloudWatchMetrics/items/properties/Metrics/items/properties/Unit",
                                            "type": "string",
                                            "title": "MetricUnit",
                                            "examples": [
                                                "Milliseconds"
                                            ],
                                            "pattern": "^(Seconds|Microseconds|
Milliseconds|Bytes|Kilobytes|Megabytes|Gigabytes|Terabytes|Bits|Kilobits|Megabits|
Gigabits|Terabits|Percent|Count|Bytes\\/Second|Kilobytes\\/Second|Megabytes\\/Second|
Gigabytes\\/Second|Terabytes\\/Second|Bits\\/Second|Kilobits\\/Second|Megabits\\/
Second|Gigabits\\/Second|Terabits\\/Second|Count\\/Second|None)$"
                                        },
                                        "StorageResolution": {
                                            "$id": "#/properties/_aws/properties/
CloudWatchMetrics/items/properties/Metrics/items/properties/StorageResolution",
                                            "type": "integer",
                                            "title": "StorageResolution",
```

```
                                        "examples": [
                                            60
                                        ]
                                    }
                                }
                            }
                        }
                    }
                }
            }
        }
    }
}
```

PutLogEvents API を使用した手動作成の埋め込みメトリクスフォーマットログの送信

CloudWatch Logs の PutLogEvents API を使用して、埋め込みメトリクスフォームログを CloudWatch Logs に送信できます。PutLogEvents を呼び出すときは、メトリクスを抽出する必要があることを CloudWatch Logs に指示するために、オプションで以下の HTTP ヘッダーを含めることができますが、必須ではなくなりました。

```
x-amzn-logs-format: json/emf
```

AWS SDK for Java 2.x を使用した詳細な例を次に示します。

```
package org.example.basicapp;

import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.cloudwatchlogs.CloudWatchLogsClient;
import software.amazon.awssdk.services.cloudwatchlogs.model.DescribeLogStreamsRequest;
import software.amazon.awssdk.services.cloudwatchlogs.model.DescribeLogStreamsResponse;
import software.amazon.awssdk.services.cloudwatchlogs.model.InputLogEvent;
import software.amazon.awssdk.services.cloudwatchlogs.model.PutLogEventsRequest;

import java.util.Collections;

public class EmbeddedMetricsExample {
        public static void main(String[] args) {
```

```
                final String usage = "To run this example, supply a Region code (eg.
 us-east-1), log group, and stream name as command line arguments"
                                + "Ex: PutLogEvents <region-id> <log-group-name>
 <stream-name>";

                if (args.length != 3) {
                        System.out.println(usage);
                        System.exit(1);
                }

                String regionId = args[0];
                String logGroupName = args[1];
                String logStreamName = args[2];

                CloudWatchLogsClient logsClient =
 CloudWatchLogsClient.builder().region(Region.of(regionId)).build();

                // Build a JSON log using the EmbeddedMetricFormat.
                long timestamp = System.currentTimeMillis();
                String message = "{" +
                                "  \"_aws\": {" +
                                "    \"Timestamp\": " + timestamp  + "," +
                                "    \"CloudWatchMetrics\": [" +
                                "      {" +
                                "        \"Namespace\": \"MyApp\"," +
                                "        \"Dimensions\": [[\"Operation\"], [\"Operation
\", \"Cell\"]]," +
                                "        \"Metrics\": [{ \"Name\": \"ProcessingLatency
\", \"Unit\": \"Milliseconds\", \"StorageResolution\": 60 }]" +
                                "      }" +
                                "    ]" +
                                "  }," +
                                "  \"Operation\": \"Aggregator\"," +
                                "  \"Cell\": \"001\"," +
                                "  \"ProcessingLatency\": 100" +
                                "}";
                InputLogEvent inputLogEvent = InputLogEvent.builder()
                        .message(message)
                        .timestamp(timestamp)
                        .build();

                // Specify the request parameters.
                PutLogEventsRequest putLogEventsRequest = PutLogEventsRequest.builder()
                        .logEvents(Collections.singletonList(inputLogEvent))
```

```
                        .logGroupName(logGroupName)
                        .logStreamName(logStreamName)
                        .build();

                logsClient.putLogEvents(putLogEventsRequest);

                System.out.println("Successfully put CloudWatch log event");
        }

}
```

> **Note**
>
> 埋め込みメトリクス形式を使用すると、アカウントの AWS/Logs 名前空間で公開されるメトリクスによって、EMF ログの処理を追跡することができます。これらのメトリクスは、EMF からのメトリクス生成の失敗を追跡するために使用できます。また、障害が発生したのは解析によるものなのか、検証によるものなのかを追跡できます。詳細については、「CloudWatch メトリクスによるモニタリング」を参照してください。

CloudWatch エージェントを使用した埋め込みメトリクスフォーマットログの送信

この方法を使用するには、まず組み込みメトリクスフォームログを送信するサービスの CloudWatch エージェントをインストールしてから、イベントの送信を開始します。

CloudWatch エージェントは、バージョン 1.230621.0 以降である必要があります。

> **Note**
>
> Lambda 関数からログを送信するために CloudWatch エージェントをインストールする必要はありません。
> Lambda 関数のタイムアウトは自動的に処理されません。つまり、メトリックがフラッシュされる前に関数がタイムアウトすると、その呼び出しのメトリックはキャプチャされません。

CloudWatch エージェントのインストール

埋め込みメトリクスフォーマットログを送信するサービスごとに CloudWatch エージェントをインストールします。

EC2 への CloudWatch エージェントのインストール

まず、インスタンスに CloudWatch エージェントをインストールします。(詳細については、CloudWatch エージェントのインストール を参照してください)。

エージェントをインストールしたら、埋め込みメトリックフォーマットログを UDP または TCP ポートでリッスンするようにエージェントを設定します。次に、デフォルトソケット `tcp:25888` をリッスンするこの設定の例を示します。エージェント設定の詳細については、CloudWatch エージェント設定ファイルを手動で作成または編集するを参照してください。

```
{
  "logs": {
    "metrics_collected": {
      "emf": { }
    }
  }
}
```

Amazon ECS への CloudWatch エージェントのインストール

Amazon ECS に CloudWatch エージェントをデプロイする最も簡単な方法は、それをサイドカーとして実行し、アプリケーションと同じタスク定義で定義することです。

エージェント設定ファイルを作成する

CloudWatch エージェント設定ファイルをローカルに作成します。この例では、相対ファイルパスは `amazon-cloudwatch-agent.json` になります。

エージェント設定の詳細については、CloudWatch エージェント設定ファイルを手動で作成または編集するを参照してください。

```
{
  "logs": {
    "metrics_collected": {
      "emf": { }
    }
```

```
  }
}
```

設定を SSM パラメータストアにプッシュする

次のコマンドを入力して、CloudWatch エージェント設定ファイルを AWS Systems Manager (SSM) Parameter Store にプッシュします。

```
aws ssm put-parameter \
    --name "cwagentconfig" \
    --type "String" \
    --value "`cat amazon-cloudwatch-agent.json`" \
    --region "{{region}}"
```

タスク定義を設定する

CloudWatch エージェントを使用して TCP または UDP ポートを公開するようにタスク定義を設定します。使用するサンプルのタスク定義は、ネットワークモードによって異なります。

webapp によって AWS_EMF_AGENT_ENDPOINT 環境変数が指定されることに注意してください。これはライブラリで使用され、エージェントがリッスンしているエンドポイントを指している必要があります。さらに、cwagent により、前の手順で作成した SSM 設定を指す「valueFrom」パラメータとして CW_CONFIG_CONTENT が指定されます。

このセクションでは、ブリッジモードの例と、ホストモードまたは awsvpc モードの例を 1 つ説明します。Amazon ECS で CloudWatch エージェントを設定する方法の例については、Github サンプルリポジトリを参照してください。

次に、ブリッジモードの例を示します。ブリッジモードネットワーキングが有効な場合、エージェントは links パラメータを使用してアプリケーションにリンクし、コンテナ名を使用してアドレス指定する必要があります。

```
{
  "containerDefinitions": [
          {
              "name": "webapp",
              "links": [ "cwagent" ],
              "image": "my-org/web-app:latest",
              "memory": 256,
              "cpu": 256,
              "environment": [{
```

```
                "name": "AWS_EMF_AGENT_ENDPOINT",
                "value": "tcp://cwagent:25888"
            }],
        },
        {
            "name": "cwagent",
            "mountPoints": [],
            "image": "public.ecr.aws/cloudwatch-agent/cloudwatch-agent:latest",
            "memory": 256,
            "cpu": 256,
            "portMappings": [{
                "protocol": "tcp",
                "containerPort": 25888
            }],
            "environment": [{
              "name": "CW_CONFIG_CONTENT",
              "valueFrom": "cwagentconfig"
            }],
        }
    ],
}
```

次に、ホストモードまたは awsvpc モードの例を示します。これらのネットワークモードで実行している場合は、localhost を介してエージェントをアドレス指定することができます。

```
{
  "containerDefinitions": [
        {
            "name": "webapp",
            "image": "my-org/web-app:latest",
            "memory": 256,
            "cpu": 256,
            "environment": [{
              "name": "AWS_EMF_AGENT_ENDPOINT",
              "value": "tcp://127.0.0.1:25888"
            }],
        },
        {
            "name": "cwagent",
            "mountPoints": [],
            "image": "public.ecr.aws/cloudwatch-agent/cloudwatch-agent:latest",
            "memory": 256,
            "cpu": 256,
```

```
                "portMappings": [{
                    "protocol": "tcp",
                    "containerPort": 25888
                }],
                "environment": [{
                  "name": "CW_CONFIG_CONTENT",
                  "valueFrom": "cwagentconfig"
                }],
            }
        ],
}
```

> **Note**
>
> awsvpc モードでは、VPC にパブリック IP アドレスを割り当てるか (Fargate のみ)、NAT ゲートウェイをセットアップするか、CloudWatch Logs VPC エンドポイントを設定する必要があります。NAT のセットアップの詳細については、「NAT ゲートウェイ」を参照してください。CloudWatch Logs VPC エンドポイントの設定の詳細については、「CloudWatch Logs とインターフェイス VPC エンドポイントの使用」を参照してください。
> 次に、Fargate 起動タイプを使用するタスクにパブリック IP アドレスを割り当てる方法の例を示します。
>
> ```
> aws ecs run-task \
> --cluster {{cluster-name}} \
> --task-definition cwagent-fargate \
> --region {{region}} \
> --launch-type FARGATE \
> --network-configuration
>  "awsvpcConfiguration={subnets=[{{subnetId}}],securityGroups=[{{sgId}}],assignPublicIp=ENA
> ```

許可を確認する

タスクを実行する IAM ロールに SSM Parameter Store から読み取るためのアクセス許可があることを確認します。このアクセス許可を追加するには、AmazonSSMReadOnlyAccess ポリシーをアタッチします。これを行うには、次のコマンドを入力します。

```
aws iam attach-role-policy --policy-arn arn:aws:iam::aws:policy/AmazonSSMReadOnlyAccess
 \
--role-name CWAgentECSExecutionRole
```

Amazon EKS への CloudWatch エージェントのインストール

このクラスターに CloudWatch Container Insights を既にインストールしている場合は、このプロセスの一部をスキップできます。

許可

Container Insights をまだインストールしていない場合は、まず Amazon EKS ノードに適切な IAM アクセス許可があることを確認してください。これらのノードに CloudWatchAgentServerPolicy がアタッチされている必要があります。(詳細については、前提条件を確認する を参照してください)。

ConfigMap を作成する

エージェントの ConfigMap を作成します。また、ConfigMap は TCP ポートまたは UDP ポートでリッスンするようにエージェントに指示します。次の ConfigMap を使用します。

```
# cwagent-emf-configmap.yaml
apiVersion: v1
data:
  # Any changes here must not break the JSON format
  cwagentconfig.json: |
    {
      "agent": {
        "omit_hostname": true
      },
      "logs": {
        "metrics_collected": {
          "emf": { }
        }
      }
    }
kind: ConfigMap
metadata:
  name: cwagentemfconfig
  namespace: default
```

Container Insights をすでにインストールしている場合は、既存の ConfigMap に次の `"emf": { }` 行を追加します。

ConfigMap を適用する

次のコマンドを入力して、ConfigMap を適用します。

```
kubectl apply -f cwagent-emf-configmap.yaml
```

エージェントをデプロイする

CloudWatch エージェントをサイドカーとしてデプロイするには、次の例のようにポッド定義にエージェントを追加します。

```
apiVersion: v1
kind: Pod
metadata:
  name: myapp
  namespace: default
spec:
  containers:
    # Your container definitions go here
    - name: web-app
      image: my-org/web-app:latest
    # CloudWatch Agent configuration
    - name: cloudwatch-agent
      image: public.ecr.aws/cloudwatch-agent/cloudwatch-agent:latest
      imagePullPolicy: Always
      resources:
        limits:
          cpu: 200m
          memory: 100Mi
        requests:
          cpu: 200m
          memory: 100Mi
      volumeMounts:
        - name: cwagentconfig
          mountPath: /etc/cwagentconfig
      ports:
  # this should match the port configured in the ConfigMap
        - protocol: TCP
          hostPort: 25888
          containerPort: 25888
  volumes:
    - name: cwagentconfig
      configMap:
        name: cwagentemfconfig
```

CloudWatch エージェントを使用した埋め込みメトリクスフォーマットログの送信

CloudWatch エージェントをインストールして実行するときに、埋め込みメトリクスフォーマットログを TCP または UDP 経由で送信できます。エージェント経由でログを送信する場合、次の 2 つの要件があります。

- ログには、どのロググループを使用するかをエージェントに指示する LogGroupName キーを含める必要があります。
- 各ログイベントは 1 行にする必要があります。つまり、ログイベントに改行 (\n) 文字を含めることはできません。

ログイベントは、埋め込みメトリックフォーマットの仕様に従う必要もあります。詳細については、「仕様: 埋め込みメトリクスフォーマット」を参照してください。

埋め込みメトリクス形式を使用して作成されたメトリクスに対してアラームを作成する予定がある場合は、「埋め込みメトリクス形式で作成されたメトリクスにアラームを設定する」の推奨事項を参照してください。

次に、Linux の bash シェルからログイベントを手動で送信する例を示します。代わりに、選択したプログラミング言語で提供される UDP ソケットインターフェイスを使用できます。

```
echo '{"_aws":{"Timestamp":1574109732004,"LogGroupName":"Foo","CloudWatchMetrics":
[{"Namespace":"MyApp","Dimensions":[["Operation"]],"Metrics":
[{"Name":"ProcessingLatency","Unit":"Milliseconds","StorageResolution":60}]}]},"Operation":"Agg
 \
> /dev/udp/0.0.0.0/25888
```

> **Note**
>
> 埋め込みメトリクス形式を使用すると、アカウントの AWS/Logs 名前空間で公開されるメトリクスによって、EMF ログの処理を追跡することができます。これらのメトリクスは、EMF からのメトリクス生成の失敗を追跡するために使用できます。また、障害が発生したのは解析によるものなのか、検証によるものなのかを追跡できます。詳細については、「CloudWatch メトリクスによるモニタリング」を参照してください。

OpenTelemetry 用の AWS ディストリビューションでの埋め込みメトリクスフォーマットの使用

埋め込みメトリックフォーマットは、OpenTelemetry プロジェクトの一部として使用できます。OpenTelemetry は、1 つの仕様と API セットを提供することにより、トレース、ログ、およびメトリクスに関するベンダー固有の形式間の境界や制限を取り除くオープンソースのイニシアチブです。詳細については、「OpenTelemetry」を参照してください。

OpenTelemetry で埋め込みメトリクスフォーマットを使用するには、OpenTelemetry に準拠しているデータソースと、CloudWatch 埋め込みメトリクスフォーマットのログで使用できる AWS Distro for OpenTelemetry の 2 つのコンポーネントが必要です。

OpenTelemetry コンポーネントの再配布は、AWS によって維持され、オンボーディングをできるだけ簡単にするために事前設定されています。埋め込みメトリクスフォーマットで OpenTelemetry を使用する方法については、他の AWS のサービスに加えて、「AWS Distro for OpenTelemetry」を参照してください。

言語サポートと使用方法の詳細については、「AWS Observability on Github」を参照してください。

コンソールでのメトリクスおよびログの表示

メトリクスを抽出する埋め込みメトリクスフォームログを生成した後、CloudWatch コンソールを使用してメトリクスを表示できます。埋め込みメトリックには、ログの生成時に指定したディメンションがあります。また、クライアントライブラリを使用して生成した埋め込みメトリックには、次のデフォルトのディメンションもあります。

- ServiceType
- ServiceName
- LogGroup

埋め込みメトリックフォーマットログから生成されたメトリックを表示するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで [Metrics] (メトリクス) を選択します。
3. 埋め込みメトリックの生成時に指定した名前空間を選択します。クライアントライブラリを使用してメトリックを生成し、名前空間を指定しなかった場合は、[aws-embedded-metrics] を選択

します。これは、クライアントライブラリを使用して生成された埋め込みメトリックのデフォルトの名前空間です。

4. メトリックディメンション ([ServiceName] など) を選択します。
5. [All metrics] タブには、名前空間内のそのディメンションのメトリクスがすべて表示されます。以下の操作を行うことができます。

   a. テーブルを並べ替えるには、列見出しを使用します。
   b. メトリクスをグラフ表示するには、メトリクスの横にあるチェックボックスを選択します。すべてのメトリクスを選択するには、テーブルの見出し行にあるチェックボックスを選択します。
   c. リソースでフィルタするには、リソース ID を選択し、[Add to search] を選択します。
   d. メトリクスでフィルタするには、メトリクスの名前を選択し、[Add to search] を選択します。

CloudWatch Logs Insights を使用したログのクエリ

CloudWatch Logs Insights を使用して、抽出されたメトリクスに関連付けられた詳細なログイベントをクエリし、運用イベントの根本原因に関する深い洞察を提供できます。ログからメトリックを抽出する利点の 1 つは、後で一意のメトリック (メトリック名と一意のディメンションセット) とメトリック値でログをフィルタリングし、集計されたメトリック値に寄与したイベントのコンテキストを取得できることです。

例えば、影響を受けたリクエスト ID または X-Ray トレース ID を取得するには、CloudWatch Logs Insights で次のクエリを実行します。

```
filter Latency > 1000 and Operation = "Aggregator"
| fields RequestId, TraceId
```

また、イベントによって影響を受けた顧客を見つけるなど、高カーディナリティキーに対してクエリ時集計を実行することもできます。以下に示しているのはこのポリシーの例です。

```
filter Latency > 1000 and Operation = "Aggregator"
| stats count() by CustomerId
```

詳細については、「CloudWatch Logs Insights を使用したログデータの分析」を参照してください

埋め込みメトリクス形式で作成されたメトリクスにアラームを設定する

通常、埋め込みメトリク形式で生成されたメトリクスに対してアラームを作成する場合、他のメトリクスにアラームを作成する場合と同じパターンに従います。詳細については、「Amazon CloudWatch でのアラームの使用」を参照してください。

埋め込みメトリクス形式のメトリクスの生成はログ公開フローによって異なります。これは、ログをメトリクスに変換するには CloudWatch Logs で処理する必要があるためです。そのため、アラームが評価される期間内にメトリクスデータポイントが作成されるように、ログを適時に公開することが重要です。

埋め込みメトリクス形式を使用して高解像度のメトリクスを送信し、そのメトリクスに対してアラームを作成する場合は、5 秒以下の間隔でログを CloudWatch Logs にフラッシュすることをおすすめします。これにより、データが一部しかない、または見つからない際に発生するアラームの原因となる追加の遅延が発生するのを避けることができます。CloudWatch エージェントを使用している場合は、CloudWatch エージェント設定ファイルで `force_flush_interval` パラメータを設定することでフラッシュ間隔を調整できます。この値のデフォルトは 5 秒です。

ログのフラッシュ間隔を制御できない他のプラットフォームで Lambda を使用している場合は、「N 中の M」アラームを使用して、アラームに使用するデータポイントの数を制御することを検討してください。詳細については、「アラームの評価」を参照してください。

CloudWatch メトリクスを発行する AWS のサービス

以下の AWS サービスは CloudWatch にメトリクスを発行します。メトリクスとディメンションについては、指定されたドキュメントを参照してください。

サービス	名前空間	ドキュメント
AWS Amplify	AWS/AmplifyHosting	モニターリング
Amazon API Gateway	AWS/ApiGateway	Amazon CloudWatch による API 実行のモニターリング
Amazon AppFlow	AWS/AppFlow	Amazon CloudWatch による Amazon AppFlow のモニターリング
AWS Application Migration Service	AWS/MGN	Amazon CloudWatch によるアプリケーション移行サービスのモニタリング
AWS App Runner	AWS/AppRunner	CloudWatch にレポートされた App Runner サービスメトリクスの表示
AppStream 2.0	AWS/AppStream	Amazon AppStream 2.0 リソースのモニターリング
AWS AppSync	AWS/AppSync	CloudWatch Metrics
Amazon Athena	AWS/Athena	CloudWatch メトリクスによる Athena クエリのモニターリング
Amazon Aurora	AWS/RDS	Amazon Aurora のメトリクス
AWS Backup	AWS/Backup	CloudWatch を使用した AWS バックアップメトリクスのモニターリング
Amazon Bedrock	AWS/Bedrock	Amazon CloudWatch による Amazon Bedrock のモニタリング

サービス	名前空間	ドキュメント
AWS Billing and Cost Management	AWS/Billing	アラートと通知による請求額のモニターリング
Amazon Braket	AWS/Braket/By Device	Amazon CloudWatch による Amazon Braket のモニタリング
AWS Certificate Manager	AWS/CertificateManager	サポートされている CloudWatch メトリクス
AWS Private CA	AWS/ACMPrivateCA	サポートされている CloudWatch メトリクス
AWS Chatbot	AWS/Chatbot	Amazon CloudWatch での AWS Chatbot のモニターリング
Amazon Chime	AWS/ChimeVoiceConnector	Amazon CloudWatch での Amazon Chime のモニタリング
Amazon Chime SDK	AWS/ChimeSDK	サービスメトリクス
AWS Client VPN	AWS/ClientVPN	Amazon CloudWatch によるモニターリング
Amazon CloudFront	AWS/CloudFront	CloudWatch を使用した CloudFront アクティビティのモニターリング
AWS CloudHSM	AWS/CloudHSM	CloudWatch メトリクスを取得する
Amazon CloudSearch	AWS/CloudSearch	Amazon CloudWatch による Amazon CloudSearch ドメインのモニターリング

サービス	名前空間	ドキュメント
AWS CloudTrail	AWS/CloudTrail	「Supported CloudWatch metrics」(サポートされている CloudWatch メトリクス)
CloudWatch エージェント	CWAgent またはカスタム名前空間	CloudWatch エージェントにより収集されるメトリクス
CloudWatch メトリクスストリーミング	AWS/CloudWatch/MetricStreams	CloudWatch メトリクスを使用したメトリクスストリームのモニターリング
CloudWatch RUM	AWS/RUM	CloudWatch RUM で収集できる CloudWatch メトリクス
CloudWatch Synthetics	CloudWatchSynthetics	Canary によって発行される CloudWatch メトリクス
Amazon CloudWatch Logs	AWS/Logs	CloudWatch メトリクスの使用状況のモニターリング
AWS CodeBuild	AWS/CodeBuild	AWS CodeBuild のモニターリング
Amazon CodeGuru Reviewer		Amazon CloudWatch での CodeGuru Reviewer のモニタリング
Amazon Kendra		Amazon CloudWatch による Amazon Kendra のモニタリング
Amazon CodeWhisperer	AWS/CodeWhisperer	Amazon CloudWatch での Amazon CodeWhisperer のモニタリング
Amazon Cognito	AWS/Cognito	Amazon Cognito のモニターリング

サービス	名前空間	ドキュメント
Amazon Comprehend	AWS/Comprehend	Amazon Comprehend エンドポイントのモニタリング
AWS Config	AWS/Config	AWS Config Usage and Success Metrics
Amazon Connect	AWS/Connect	Amazon CloudWatch メトリクスでの Amazon Connect のモニターリング
Amazon Data Lifecycle Manager	AWS/DataLifecycleManager	Amazon CloudWatch を使用してのポリシーのモニターリング
AWS DataSync	AWS/DataSync	タスクのモニターリング
Amazon DataZone		Amazon CloudWatch による Amazon DataZone のモニタリング
Amazon DevOps Guru	AWS/DevOps-Guru	Amazon CloudWatch での Amazon DevOps Guru のモニタリング
AWS Database Migration Service	AWS/DMS	AWS DMS タスクのモニターリング
AWS Direct Connect	AWS/DX	Amazon CloudWatch によるモニターリング
AWS Directory Service	AWS/DirectoryService	Amazon CloudWatch メトリクスを使用して、ドメインコントローラーを追加するタイミングを決定する
Amazon DocumentDB	AWS/DocDB	Amazon DocumentDB のメトリクス
Amazon DynamoDB	AWS/DynamoDB	DynamoDB のメトリクスとディメンション

サービス	名前空間	ドキュメント
DynamoDB Accelerator (DAX)	AWS/DAX	DAX メトリクスおよびディメンションの表示
Amazon EC2	AWS/EC2	CloudWatch を使用したインスタンスのモニターリング
Amazon EC2 Elastic Graphics	AWS/ElasticGPUs	CloudWatch メトリクスを使用した Elastic Graphics のモニターリング
Amazon EC2 スポットフリート	AWS/EC2Spot	スポットフリートの CloudWatch メトリクス
Amazon EC2 Auto Scaling (日本語)	AWS/AutoScaling	CloudWatch を使用した Auto Scaling グループとインスタンスのモニターリング
AWS Elastic Beanstalk	AWS/ElasticBeanstalk	環境の Amazon CloudWatch カスタムメトリクスの発行
Amazon Elastic Block Store	AWS/EBS	Amazon EBS の Amazon CloudWatch メトリクス
Amazon Elastic Container Registry	AWS/ECR	Amazon ECR リポジトリメトリクス
Amazon Elastic Container Service	AWS/ECS	Amazon ECS CloudWatch メトリクス
CloudWatch Container Insights を介した Amazon ECS	ECS/ContainerInsights	Amazon ECS Container Insights メトリクス

サービス	名前空間	ドキュメント
Amazon ECS クラスターの Auto Scaling	AWS/ECS/ManagedScaling	Amazon ECS クラスターの Auto Scaling
AWS Elastic Disaster Recovery		DRS の CloudWatch メトリクス
Amazon Elastic File System	AWS/EFS	CloudWatch によるモニターリング
Amazon Elastic Inference	AWS/ElasticInference	Using CloudWatch Metrics to Monitor Amazon Elastic Inference
CloudWatch Container Insights を介した Amazon EKS	ContainerInsights	Amazon EKS および Kubernetes Container Insights のメトリクス
Elastic Load Balancing	AWS/ApplicationELB	Application Load Balancer の CloudWatch メトリクス
Elastic Load Balancing	AWS/NetworkELB	Network Load Balancer の CloudWatch メトリクス
Elastic Load Balancing	AWS/GatewayELB	Gateway Load Balancer の CloudWatch メトリクス
Elastic Load Balancing	AWS/ELB	Classic Load Balancer の CloudWatch メトリクス
Amazon Elastic Transcoder	AWS/ElasticTranscoder	Amazon CloudWatch によるモニターリング

サービス	名前空間	ドキュメント
Amazon ElastiCache for Memcached	AWS/ElastiCache	CloudWatch メトリクスの使用状況のモニターリング
Amazon ElastiCache for Redis	AWS/ElastiCache	CloudWatch メトリクスの使用状況のモニターリング
Amazon OpenSearch Service	AWS/ES	Amazon CloudWatch を用いた OpenSearch クラスターメトリクスのモニターリング
Amazon EMR	AWS/ElasticMapReduce	CloudWatch によるメトリクスのモニターリング
AWS Elemental MediaConnect	AWS/MediaConnect	Amazon CloudWatch による MediaConnect のモニターリング
AWS Elemental MediaConvert	AWS/MediaConvert	CloudWatch メトリクスを使用した AWS Elemental MediaConvert リソースに関するメトリクスの表示
AWS Elemental MediaLive	AWS/MediaLive	Amazon CloudWatch メトリクスを使用してのアクティビティのモニターリング
AWS Elemental MediaPackage	AWS/MediaPackage	Amazon CloudWatch メトリクスによる AWS Elemental MediaPackage のモニタリング
AWS Elemental MediaStore	AWS/MediaStore	Amazon CloudWatch メトリクスによる AWS Elemental MediaStore のモニタリング
AWS Elemental MediaTailor	AWS/MediaTailor	Amazon CloudWatch での AWS Elemental MediaTailor のモニターリング
Amazon EventBridge	AWS/Events	Amazon EventBridge のモニターリング

サービス	名前空間	ドキュメント
Amazon FinSpace		ログ記録とモニタリング
Amazon Forecast		Amazon Forecast 用 CloudWatch メトリクス
Amazon Fraud Detector		Amazon CloudWatch を使用した Amazon Fraud Detector のモニタリング
Amazon FSx for Lustre	AWS/FSx	Amazon FSx for Lustre のモニターリング
Amazon FSx for OpenZFS	AWS/FSx	Amazon CloudWatch によるモニタリング
Amazon FSx for Windows File Server	AWS/FSx	Amazon FSx for Windows File Server のモニターリング
Amazon FSx for NetApp ONTAP	AWS/FSx	Amazon CloudWatch によるモニターリング
Amazon FSx for OpenZFS	AWS/FSx	Amazon CloudWatch によるモニタリング
Amazon GameLift	AWS/GameLift	CloudWatch による Amazon GameLift のモニターリング
AWS Global Accelerator	AWS/GlobalAccelerator	AWS Global Accelerator での Amazon CloudWatch アラームの使用
AWS Glue	Glue	CloudWatch メトリクスを使用した AWS Glue のモニターリング
AWS Ground Station	AWS/GroundStation	Amazon CloudWatch を使用したメトリクス

サービス	名前空間	ドキュメント
AWS HealthLake	AWS/HealthLake	CloudWatch による HealthLake のモニターリング
Amazon Inspector	AWS/Inspector	CloudWatch を使用した Amazon Inspector のモニターリング
Amazon Interactive Video Service	AWS/IVS	Amazon CloudWatch による Amazon IVS のモニターリング
Amazon Interactive Video Service Chat	AWS/IVSChat	Amazon CloudWatch による Amazon IVS のモニターリング
AWS IoT	AWS/IoT	AWS IoT のメトリクスとディメンション
AWS IoT Analytics	AWS/IoTAnalytics	メトリクス、ディメンション、名前空間
AWS IoT FleetWise	AWS/IoTFleetWise	Amazon CloudWatch による AWS IoT FleetWise のモニターリング
AWS IoT SiteWise	AWS/IoTSiteWise	Amazon CloudWatch メトリクスによる AWS IoT SiteWise のモニターリング
AWS IoT TwinMaker	AWS/IoTTwinMaker	Amazon CloudWatch メトリクスによる AWS IoT TwinMaker のモニターリング
AWS IoT 1-Click		Amazon CloudWatch を使用した AWS IoT 1-Click のモニタリング
AWS Key Management Service	AWS/KMS	CloudWatch によるモニターリング

サービス	名前空間	ドキュメント
Amazon Keyspaces (Apache Cassandra 用)	AWS/Cassandra	Amazon Keyspaces のメトリクスとディメンション
Amazon Kendra		Amazon CloudWatch による Amazon Kendra のモニタリング
Amazon Managed Service for Apache Flink	AWS/KinesisAnalytics	SQL アプリケーション用 Managed Service for Apache Flink:: CloudWatch によるモニタリング Managed Service for Apache Flink for Apache Flink: Amazon Managed Service for Apache Flink のメトリクスとディメンションの表示
Amazon Data Firehose	AWS/Firehose	Monitoring Amazon Data Firehose Using CloudWatch Metrics
Amazon Kinesis Data Streams	AWS/Kinesis	Amazon CloudWatch による Amazon Kinesis Data Streams のモニターリング
Amazon Kinesis Video Streams	AWS/KinesisVideo	CloudWatch による Kinesis Video Streams Metrics のモニターリング
AWS Lambda	AWS/Lambda	AWS Lambda のメトリクス
Amazon Lex	AWS/Lex	Amazon CloudWatch による Amazon Lex のモニターリング

サービス	名前空間	ドキュメント

AWS License Manager	`AWSLicenseManager/licenseUsage`
	
	`AWS/LicenseManager/LinuxSubscriptions`
		Amazon CloudWatch を使用したライセンス使用状況のモニタリング
		
		Linux サブスクリプションの使用状況メトリクスと Amazon CloudWatch アラーム

Amazon Location Service	`AWS/Location`	Amazon CloudWatch にエクスポートされた Amazon Location Service のメトリクス
Amazon Lookout for Equipment	`AWS/lookoutequipment`	Amazon CloudWatch による Lookout for Equipment のモニタリング
Amazon Lookout for Metrics	`AWS/LookoutMetrics`	Amazon CloudWatch による Lookout for Metrics のモニターリング
Amazon Lookout for Vision	`AWS/LookoutVision`	Amazon CloudWatch による Lookout for Vision のモニタリング
AWS Mainframe Modernization		Amazon CloudWatch による AWS メインフレームモダナイゼーションのモニタリング
Amazon Machine Learning	`AWS/ML`	Amazon ML と CloudWatch メトリクスのモニターリング
Amazon Managed Blockchain	`AWS/managedblockchain`	Amazon Managed Blockchain で Hyperledger Fabric Peer ノードメトリクスを使用

サービス	名前空間	ドキュメント
Amazon Managed Service for Prometheus	AWS/Prometheus	Amazon CloudWatch メトリクス
Amazon Managed Streaming for Apache Kafka	AWS/Kafka	Amazon CloudWatch による Amazon MSK のモニターリング
Amazon Managed Streaming for Apache Kafka	AWS/KafkaConnect	MSK Connect のモニタリング
Amazon Managed Workflows for Apache Airflow	AWS/MWAA	Container, queue, and database metrics for Amazon MWAA
Amazon MemoryDB for Redis	AWS/MemoryDB	CloudWatch メトリクスを使用したモニタリング
Amazon MQ	AWS/AmazonMQ	Amazon CloudWatch を使用した Amazon MQ ブローカーのモニターリング
Amazon Neptune	AWS/Neptune	CloudWatch による Neptune のモニターリング
AWS Network Firewall	AWS/NetworkFirewall	Amazon CloudWatch の AWS Network Firewall メトリクス
AWS Network Manager	AWS/NetworkManager	オンプレミスリソースの CloudWatch メトリクス

サービス	名前空間	ドキュメント
Amazon Nimble Studio	AWS/NimbleStudio	Amazon CloudWatch で Nimble Studio をモニターリングする
AWS HealthOmics	AWS/Omics	Amazon CloudWatch での AWS HealthOmics のモニタリング
AWS OpsWorks	AWS/OpsWorks	Amazon CloudWatch を使用した Stacks のモニターリング
AWS Outposts	AWS/Outposts	AWS Outposts の CloudWatch メトリクス
AWS Panorama	AWS/PanoramaDeviceMetrics	Amazon CloudWatch によるアプライアンスとアプリケーションのモニタリング
Amazon Personalize	AWS/Personalize	Amazon Personalize の CloudWatch メトリクス
Amazon Pinpoint	AWS/Pinpoint	CloudWatch で Amazon Pinpoint メトリクスを表示する
Amazon Polly	AWS/Polly	CloudWatch と Amazon Polly の統合
AWS PrivateLink	AWS/PrivateLinkEndpoints	AWS PrivateLink の CloudWatch メトリクス
AWS PrivateLink	AWS/PrivateLinkServices	AWS PrivateLink の CloudWatch メトリクス
AWS Private 5G	AWS/Private5G	Amazon CloudWatch メトリクス
Amazon QLDB	AWS/QLDB	Amazon QuickSight でのデータのモニターリング

サービス	名前空間	ドキュメント
Amazon QuickSight	AWS/QuickSight	Amazon CloudWatch によるモニターリング
Amazon Redshift	AWS/Redshift	Amazon Redshift パフォーマンスデータ
Amazon Relational Database Service	AWS/RDS	Amazon CloudWatch を使用した Amazon RDS メトリクスのモニターリング
Amazon Rekognition	AWS/Rekognition	Amazon CloudWatch による Rekognition のモニタリング
AWS re:Post Private	AWS/rePostPrivate	Amazon CloudWatch による AWS re:Post Private のモニタリング
AWS RoboMaker	AWS/Robomaker	Amazon CloudWatch による AWS RoboMaker のモニターリング
Amazon Route 53	AWS/Route53	Amazon Route 53 のモニターリング
Route 53 Application Recovery Controller	AWS/Route53RecoveryReadiness	Amazon CloudWatch と Application Recovery Controller の使用
Amazon SageMaker	AWS/SageMaker	CloudWatch による SageMaker のモニターリング
Amazon SageMaker Model Building Pipelines	AWS/SageMaker/ModelBuildingPipeline	SageMaker パイプラインメトリクス

サービス	名前空間	ドキュメント
AWS Secrets Manager	AWS/SecretsManager	Amazon CloudWatch による Secrets Manager のモニターリング
Amazon Security Lake	AWS/SecurityLake	Amazon Security Lake の CloudWatch メトリクス
Service Catalog	AWS/ServiceCatalog	Service Catalog の CloudWatch メトリクス
AWS Shield Advanced	AWS/DDoSProtection	CloudWatch によるモニターリング
Amazon Simple Email Service	AWS/SES	CloudWatch からの Amazon SES イベントデータの取得
AWS SimSpace Weaver	AWS/simspaceweaver	Amazon CloudWatch での AWS SimSpace Weaver のモニタリング
Amazon Simple Notification Service	AWS/SNS	CloudWatch による Amazon SNS のモニターリング
Amazon Simple Queue Service	AWS/SQS	CloudWatch を使用した Amazon SQS キューのモニターリング
Amazon S3	AWS/S3	Amazon CloudWatch によるメトリクスのモニターリング
S3 Storage Lens	AWS/S3/Storage-Lens	CloudWatch の S3 Storage Lens メトリクスのモニターリング
Amazon Simple Workflow Service	AWS/SWF	CloudWatch の Amazon SWF メトリクス
AWS Step Functions	AWS/States	CloudWatch を使用した Step Functions のモニターリング

サービス	名前空間	ドキュメント
AWS Storage Gateway	AWS/StorageGateway	Amazon CloudWatch メトリクスを使用する
AWS Systems Manager Run Command	AWS/SSM-RunCommand	CloudWatch を使用した Run Command メトリクスのモニターリング
Amazon Textract	AWS/Textract	Amazon Textract 用の CloudWatch メトリクス
Amazon Timestream	AWS/Timestream	Timestream のメトリクスとディメンション
AWS Transfer for SFTP	AWS/Transfer	AWS SFTP CloudWatch メトリクス
Amazon Transcribe	AWS/Transcribe	Amazon CloudWatch での Amazon Transcribe のモニタリング
Amazon Translate	AWS/Translate	Amazon Translate の CloudWatch メトリクスとディメンション
AWS Trusted Advisor	AWS/TrustedAdvisor	CloudWatch を使用した Trusted Advisor アラームの作成
Amazon VPC	AWS/NATGateway	Amazon CloudWatch で NAT ゲートウェイを監視する
Amazon VPC	AWS/TransitGateway	Transit Gateways 用の CloudWatch メトリクス
Amazon VPC	AWS/VPN	CloudWatch によるモニターリング
Amazon VPC IP Address Manager	AWS/IPAM	Amazon CloudWatch アラームの作成

サービス	名前空間	ドキュメント
AWS WAF	AWS WAF リソース用の `AWS/WAFV2` AWS WAF クラシックリソース用の `WAF`	CloudWatch によるモニターリング
Amazon WorkMail	`AWS/WorkMail`	Amazon CloudWatch での Amazon WorkMail のモニタリング
Amazon WorkSpaces	`AWS/WorkSpaces`	CloudWatch メトリクスを使用した WorkSpaces のモニターリング
Amazon WorkSpaces Web	`AWS/WorkSpacesWeb`	Amazon CloudWatch による Amazon WorkSpaces Web のモニターリング

AWS 使用状況メトリクス

CloudWatch は、一部の AWS リソースおよび API の使用状況を追跡するメトリクスを収集します。これらのメトリクスは、AWS/Usage 名前空間内で発行されます。CloudWatch の使用状況メトリクスを使用して、使用状況をプロアクティブに管理することができます。これは、CloudWatch コンソールでのメトリクスの可視化、カスタムダッシュボードの作成、CloudWatch 異常検出によるアクティビティの変化の検出、使用量がしきい値に近づいたときに警告するアラームの設定などによって実現します。

一部の AWS サービスは、これらの使用状況メトリクスを Service Quotas と統合します。これらのサービスでは、CloudWatch を使用して、アカウントのサービスクォータの使用を管理できます。詳細については、「サービスクォータの可視化とアラームの設定」を参照してください。

トピック

- サービスクォータの可視化とアラームの設定
- AWS API 使用状況メトリクス
- CloudWatch の使用状況メトリクス

サービスクォータの可視化とアラームの設定

一部の AWS サービスでは、これらの使用状況メトリクスを使用して、CloudWatch グラフやダッシュボードで現在のサービスの使用状況を可視化できます。CloudWatch のメトリクスの数学関数を使用すると、それらリソースのサービスクォータをグラフ化できます。また、使用量がサービスクォータに近づいたときに警告するアラームも設定できます。サービスクォータの詳細については、Service Quotas ユーザーガイドの Service Quotas とは を参照してください。

CloudWatch のクロスアカウントオブザーバビリティでモニタリングアカウントとして設定されたアカウントにサインインしている場合、そのモニタリングアカウントを使用して、Service Quotas を視覚化し、そのモニタリングアカウントにリンクされているソースアカウントのメトリクスのアラームを設定できます。詳細については、「CloudWatch のクロスアカウントオブザーバビリティ」を参照してください。

現在、次のサービスは、使用状況メトリクスを Service Quotas と統合しています。

- AWS CloudHSM
- Amazon Chime SDK

- 「Amazon CloudWatch」
- Amazon CloudWatch Logs
- Amazon DynamoDB
- Amazon EC2
- Amazon Elastic Container Registry
- AWS Fargate
- AWS Fault Injection Service
- AWS Interactive Video Service
- AWS Key Management Service
- Amazon Kinesis Data Firehose
- Amazon Location Service
- Amazon Managed Blockchain (AMB) Query
- AWS Robomaker
- Amazon SageMaker

サービスクォータを可視化し、オプションでアラームを設定するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで [Metrics (メトリクス)] を選択します。
3. [All metrics] (すべてのメトリクス) タブで [Usage] (使用状況) をクリックした後、[By AWS Resource] (AWS リソース別) をクリックします。

   サービスクォータ使用状況メトリクスのリストが表示されます。
4. いずれかのメトリクスの横にあるチェックボックスをオンにします。

   グラフには、その AWS リソースの現在の使用状況が表示されます。
5. サービスクォータをグラフに追加するには、次の手順を実行します。

   a. [グラフ化したメトリクス] タブを選択します。

   b. [Math expression (数式)]、[Start with an empty expression (空の式で開始)] の順に選択します。新しい行の [Details] (詳細) で **`SERVICE_QUOTA(m1)`** と入力します。

      グラフに新しい行が追加され、メトリクスで表されるリソースのサービスクォータが表示されます。

6. 現在の使用状況をクォータの割合として表示するには、新しい式を追加するか、現在の [SERVICE_QUOTA] 式を変更します。使用する新しい式は **`m1/SERVICE_QUOTA(m1)*100`** となります。
7. （オプション）サービスクォータに近づいた場合に通知するアラームを設定するには、次の手順を実行します。

   a. **`m1/SERVICE_QUOTA(m1)*100`** を含む行の [Actions] (アクション) で、アラームアイコンをクリックします。それはベルのように見えます。

      アラーム作成ページが表示されます。

   b. [Conditions (条件)] で、[Threshold type (しきい値の種類)] が [Static (静的)] で、[Whenever Expression1 is (式 1)] が [Greater (大きい)] に設定されていることを確認します。[than (以上)] に「**`80`**」と入力します。これにより、使用量がクォータの 80% を超えたときに ALARM 状態になるアラームが作成されます。

   c. [Next] を選択します。

   d. 次のページで、Amazon SNS トピックを選択するか、新しいトピックを作成し、[Next] (次へ) を選択します。選択するトピックは、アラームが ALARM 状態になったときに通知されます。

   e. 次のページで、アラームの名前と説明を入力し、[Next (次へ)] を選択します。

   f. アラームの作成 を選択します。

AWS API 使用状況メトリクス

AWS CloudTrail ログ記録をサポートするほとんどの API は、CloudWatch に使用状況メトリクスもレポートします。CloudWatch の API 使用状況メトリクスを使用して、API の使用状況をプロアクティブに管理することができます。これは、CloudWatch コンソールでのメトリクスの可視化、カスタムダッシュボードの作成、CloudWatch 異常検出によるアクティビティの変化の検出、使用量がしきい値に近づいたときに警告するアラームの設定などによって実現します。

次の表は、CloudWatch に API の使用状況メトリクスをレポートするサービスと、そのサービスの使用状況メトリクスを表示するために `Service` ディメンションに使用する値の一覧です。

サービス	**Service** ディメンションの値
AWS Identity and Access Management Access Analyzer	`Access Analyzer`
AWS Account Management	`Account Management`
Alexa for Business	`A4B`
Amazon API Gateway	`API Gateway`
AWS App Mesh	`App Mesh`
AWS AppConfig	`AWS AppConfig`
Amazon AppFlow	`AppFlow`
Application Auto Scaling	`Application Auto Scaling`
Application Discovery Service	`Application Discovery Service`
Amazon AppStream	`AppStream`
AppStream 2.0 Image Builder	`Image Builder`
Amazon Athena	`Athena`
AWS Audit Manager	`Audit Manager`
AWS Backup	`Backup`
AWS Batch	`Batch`
Amazon Braket	`Braket`
AWS Budgets	`Budgets`
AWS Certificate Manager	`Certificate Manager`
Amazon Chime SDK	`ChimeSDK`

サービス	**Service** ディメンションの値
Amazon Cloud Directory	`Cloud Directory`
AWS Cloud Map	`Cloud Map`
AWS CloudFormation	`CloudFormation`
AWS CloudHSM	`CloudHSM`
Amazon CloudSearch	`CloudSearch`
AWS CloudShell	`CloudShell`
AWS CloudTrail	`CloudTrail`
Amazon CloudWatch	`CloudWatch`
Amazon CloudWatch Logs	`Logs`
Amazon CloudWatch Application Insights	`CloudWatch Application Insights`
AWS CodeBuild	`CodeBuild`
AWS CodeCommit	`CodeCommit`
Amazon CodeGuru Profiler	`CodeGuru Profiler`
AWS CodePipeline	`CodePipeline`
AWS CodeStar	`CodeStar`
AWS CodeStar 通知	`CodeStar Notifications`
AWS CodeStar Connections	`CodeStar Connections`
Amazon Cognito アイデンティティプール	`Cognito Identity Pools`
Amazon Cognito Sync	`Cognito Sync`
Amazon Comprehend	`Comprehend`

サービス	**Service** ディメンションの値
Amazon Comprehend Medical	`Comprehend Medical`
AWS Compute Optimizer	`ComputeOptimzier`
Amazon Connect	`Connect`
Amazon Connect Customer Profiles	`Customer Profiles`
AWS コストと使用状況レポート	`Cost and Usage Report`
AWS Cost Explorer	`Cost Explorer`
AWS Data Exchange	`Data Exchange`
AWS Data Lifecycle Manager	`Data Lifecycle Manager`
AWS Database Migration Service	`Database Migration Service`
AWS DataSync	`DataSync`
AWS DeepLens	`AWS DeepLens`
Amazon Detective	`Detective`
デバイスアドバイザー	`Device Advisor`
AWS Direct Connect	`Direct Connect`
AWS Directory Service	`Directory Service`
DynamoDB Accelerator	`DynamoDBAccelerator`
Amazon EC2	`EC2`
EC2 オートスケーリング	`EC2 Auto Scaling`
Amazon Elastic Container Registry	`ECR Public`
Amazon Elastic Container Service	`ECS`

サービス	**Service** ディメンションの値
Amazon Elastic File System	EFS
Amazon Elastic Kubernetes Service	EKS
AWS Elastic Beanstalk	Elastic Beanstalk
Amazon Elastic Inference	Elastic Inference
Elastic Load Balancing	Elastic Load Balancing
Amazon EMR	EMR Containers
AWS Firewall Manager	Firewall Manager
Amazon FSx	FSx
Amazon GameLift	GameLift
AWS Glue DataBrew	DataBrew
Amazon Managed Grafana	Grafana
AWS IoT Greengrass	Greengrass
AWS Ground Station	Ground Station
AWS Health API と通知	AWS Health APIs And Notifications
Amazon Interactive Video Service	IVS
AWS IoT Core	IoT
AWS IoT 1-Click	IoT 1-Click
AWS IoT Events	IoT Events
AWS IoT RoboRunner	IoT RoboRunner
AWS IoT SiteWise	IoT Sitewise

サービス	**Service** ディメンションの値
AWS IoT Wireless	`IoT Wireless`
Amazon Kendra	`Kendra`
Amazon Keyspaces (Apache Cassandra 向け)	`Keyspaces`
Amazon Managed Service for Apache Flink	`Kinesis Analytics`
Amazon Kinesis Data Firehose	`Firehose`
Kinesis Video Streams	`Kinesis Video Streams`
AWS Key Management Service	`KMS`
AWS Lambda	`Lambda`
AWS Launch Wizard	`Launch Wizard`
Amazon Lex	`Amazon Lex`
Amazon Lightsail	`Lightsail`
Amazon Location Service	`Location`
Amazon Lookout for Vision	`Lookout for Vision`
Amazon Machine Learning	`Amazon Machine Learning`
Amazon Macie	`Macie`
Amazon Managed Blockchain (AMB) Query	`Amazon Managed Blockchain Query`
AWS Managed Services	`AWS Managed Services`
AWS Marketplace Commerce Analytics	`Marketplace Analytics Service`
AWS Elemental MediaConnect	`MediaConnect`
AWS Elemental MediaConvert	`MediaConvert`

サービス	**Service** ディメンションの値
AWS Elemental MediaLive	`MediaLive`
AWS Elemental MediaStore	`Mediastore`
AWS Elemental MediaTailor	`MediaTailor`
AWS Mobile Hub	`Mobile Hub`
AWS Network Firewall	`Network Firewall`
AWS OpsWorks	`OpsWorks`
AWS OpsWorks設定管理のための	`OPsWorks CM`
AWS Outposts	`Outposts`
AWS Organizations	`Organizations`
「Amazon RDS Performance Insights」	`Performance Insights`
Amazon Pinpoint	`Pinpoint`
AWS Private Certificate Authority	`Private Certificate Authority`
Amazon Managed Service for Prometheus	`Prometheus`
AWS Proton	`Proton`
Amazon Quantum Ledger Database (Amazon QLDB)	`QLDB`
Amazon RDS	`RDS`
Amazon Redshift	`Redshift Data API`
Amazon Rekognition	`Rekognition`
AWS Resource Access Manager	`Resource Access Manager`
AWS Resource Groups	`Resource Groups`

サービス	**Service** ディメンションの値
AWS Resource Groups Tagging API	`Resource Groups Tagging API`
AWS RoboMaker	`RoboMaker`
Amazon Route 53 Domains	`Route 53 Domains`
Amazon Route 53 Resolver	`Route 53 Resolver`
Amazon S3	`S3`
Amazon S3 Glacier	`Amazon S3 Glacier`
Amazon SageMaker ランタイム	`Sagemaker`
Savings Plans	`Savings Plans`
AWS Secrets Manager	`Secrets Manager`
AWS Security Hub	`Security Hub`
AWS Server Migration Service	`AWS Server Migration Service`
AWS Service Catalog AppRegistry	`Service Catalog AppRegistry`
Service Quotas	`Service Quotas`
AWS Shield	`Shield`
AWS Signer	`Signer`
Amazon Simple Notification Service	`SNS`
Amazon Simple Email Service	`SES`
Amazon Simple Queue Service	`SQS`
ID ストア	`Identity Store`
Storage Gateway	`Storage Gateway`

サービス	**Service** ディメンションの値
AWS Support	`Support`
Amazon Simple Workflow Service	`SWF`
Amazon Textract	`Textract`
AWS IoT Things Graph	`ThingsGraph`
Amazon Timestream	`Timestream`
Amazon Transcribe	`Transcribe`
Amazon Translate	`Translate`
Amazon Transcribe ストリーミングトランスクリプション	`Transcribe Streaming`
AWS Transfer Family	`Transfer`
AWS WAF	`WAF`
Amazon WorkDocs	`Amazon WorkDocs`
Amazon WorkLink	`WorkLink`
Amazon WorkMail	`Amazon WorkMail`
Amazon WorkSpaces	`Workspaces`
AWS X-Ray	`X-Ray`

一部のサービスでは、追加の API の使用状況メトリクスもレポートされます。API が CloudWatch に使用状況メトリクスをレポートするかどうかを確認するには、CloudWatch コンソールを使用して、`AWS/Usage` 名前空間内のそのサービスによってレポートされたメトリクスを確認します。

CloudWatch に使用状況のメトリクスをレポートするサービスの API のリストを表示するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。

2. ナビゲーションペインで [Metrics (メトリクス)] を選択します。
3. [All metrics] (すべてのメトリクス) タブで [Usage] (使用状況) をクリックした後、[By AWS Resource] (AWS リソース別) をクリックします。
4. メトリクスのリストの近くの検索ボックスに、サービスの名前を入力します。メトリクスは、入力したサービスによってフィルタリングされます。

CloudWatch の使用状況メトリクス

CloudWatch は、一部の AWS リソースの使用状況を追跡するメトリクスを収集します。これらのメトリクスは、AWS のサービスクォータに対応しています。これらのメトリクスを追跡することで、クォータを積極的に管理できます。詳細については、「サービスクォータの可視化とアラームの設定」を参照してください。

サービスクォータの使用状況メトリクスは AWS/Usage 名前空間にあり、1 分ごとに収集されます。

現在、この名前空間で CloudWatch が発行する唯一のメトリクス名は CallCount です。このメトリクスは、Resource、Service、および Type のディメンションで発行されます。Resource ディメンションは、追跡される API オペレーションの名前を指定します。例えば、ディメンションが "Service": "CloudWatch"、"Type": "API"、および "Resource": "PutMetricData" の CallCount メトリクスは、ユーザーのアカウントで CloudWatch PutMetricData API オペレーションが呼び出された回数を示します。

CallCount メトリクスには指定された単位がありません。メトリクスの最も有用な統計は SUM です。これは、1 分間の合計オペレーション数を表します。

メトリクス

メトリクス	説明
CallCount	アカウントで実行された指定されたオペレーションの数。

[Dimensions] (ディメンション):

ディメンション	説明
Service	リソースを含む AWS のサービスの名前。CloudWatch 使用状況メトリクスの場合、このディメンションの値は CloudWatch です。

ディメンション	説明
Class	追跡されているリソースのクラス。CloudWatch API 使用状況メトリクスでは、値が None のこのディメンションを使用します。
Type	追跡されるリソースのタイプ。現在、Service ディメンションが CloudWatch である場合、Type の有効な値は API のみです。
Resource	API オペレーションの名前。有効な値には次のようなものがあります。DeleteAlarms 、DeleteDashboards 、DescribeAlarmHistory 、DescribeAlarms 、GetDashboard 、GetMetricData 、GetMetricStatistics 、ListMetrics 、PutDashboard 、および PutMetricData

CloudWatch チュートリアル

以下のシナリオは、Amazon CloudWatch の使用方法を示しています。最初のシナリオでは、CloudWatch コンソールを使用して、AWS の使用量を追跡して一定の消費のしきい値を超えたときに通知する、請求アラームを作成します。2 番目のより高度なシナリオでは、AWS Command Line Interface (AWS CLI) を使用し、GetStarted という仮想アプリケーションの 1 つのメトリクスを発行します。

シナリオ

- 予想請求額のモニターリング
- メトリクスの発行

シナリオ: CloudWatch で予想請求額をモニターリングする

このシナリオでは、予想請求額をモニターリングする Amazon CloudWatch アラームを作成します。ご使用の AWS アカウントに対する予想請求額のモニターリングを有効にすると、予想請求額が計算されて、メトリクスデータとして CloudWatch に 1 日複数回送信されます。

請求メトリックスデータは、米国東部 (バージニア北部) リージョンに保存され、全世界の料金が反映されます。このデータには、使用した AWS のサービス別の予想請求額と、AWS 全体の予想請求額の合計が含まれます。

請求額が所定のしきい値を超過したときにメールでアラートを受け取ることができます。これらのアラートは CloudWatch によってトリガーされ、Amazon Simple Notification Service (Amazon SNS) の通知を用いて送信されます。

> **Note**
>
> 既に請求されている CloudWatch 料金の分析については、「CloudWatch の請求とコスト」を参照してください。

タスク

- ステップ 1: 請求アラートを有効にする
- ステップ 2: 請求アラームを作成する
- ステップ 3: アラームの状態をチェックする

ステップ 1: 請求アラートを有効にする

予想請求額のアラームを作成する前に、請求アラートを有効にする必要があります。有効にすると、AWS の予想請求額をモニターリングし、請求メトリクスデータを使用してアラームを作成できます。請求アラートを有効にすると、データの収集を無効にできなくなりますが、作成した請求アラームは削除できます。

初めて予想請求額のモニターリングを有効にした場合、請求データの表示と請求アラートの設定ができるようになるまで約 15 分かかります。

要件

- ルートユーザー認証情報を使用するか、請求情報を表示するアクセス許可が付与されたユーザーとして、サインインする必要があります。
- 一括請求 (コンソリデーティッドビリング) のアカウントの場合、支払いアカウントでログインすると、リンクされている各アカウントの請求データを見ることができます。リンクされているそれぞれのアカウントと一括請求アカウントのどちらに対しても、予想請求合計額とサービスごとの予想請求額のデータを見ることができます。
- 一括請求 (コンソリデーティッドビリング) アカウントでは、メンバー連結アカウントのメトリクスは、支払者アカウントが [請求アラートを受け取る] 設定を有効にしている場合にのみキャプチャされます。管理/支払者アカウントのアカウントを変更する場合は、新しい管理/支払者アカウントの請求アラートを有効にする必要があります。
- Amazon パートナーネットワーク (APN) アカウントの請求メトリクスは CloudWatch に対して公開されないため、このアカウントを APN の一部にすることはできません。詳細については、「AWS パートナーネットワーク」を参照してください。

予想請求額のモニターリングを有効にするには

1. AWS Billing コンソール https://console.aws.amazon.com/billing/ を開きます。
2. ナビゲーションペインで、[請求設定] を選択します。
3. [アラート設定] で [編集] を選択します。
4. [CloudWatch 請求アラートを受信する] を選択します。
5. [詳細設定を保存] を選択します。

ステップ 2: 請求アラームを作成する

> **Important**
>
> 請求アラームを作成する前に、リージョンを米国東部 (バージニア北部) に設定する必要があります。請求メトリクスデータは、このリージョンに保存され、世界全体の請求額を示します。また、自分のアカウントまたは管理/支払者アカウント (一括請求を使用している場合) で、請求アラートを有効にする必要があります。詳細については、「ステップ 1: 請求アラートを有効にする」を参照してください。

この手順では、AWS の予想請求額が、定義されたしきい値を超えた場合に通知を送信するアラームを作成します。

CloudWatch コンソールを用いて請求アラームを作成するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで、[Alarms] (アラーム) を選択し、[All alarms] (アラームの作成) を選択します。
3. [アラームの作成] を選択します。
4. [Select metric] (メトリクスの選択) を選択します。[Browse] (参照) で、[Billing] (請求) を選択してから、[Total Estimated Charge] (概算合計請求額) を選択します。

   > **Note**
   >
   > [請求]/[概算合計請求額] メトリクスが表示されない場合は、請求アラートを有効にして、リージョンを米国東部 (バージニア北部) に変更します。詳細については、「請求アラートの有効化」を参照してください。

5. [EstimatedCharges] メトリクスの横にあるチェックボックスをオンにして、[Select metric] (メトリクスの選択) を選択します。
6. [統計] で、[Maximum] を選択します。
7. [Period] (期間) で、[6 hours] (6 時間) を選択します。
8. [Threshold type] で [静的] を選択します。
9. [Whenever EstimatedCharges is . . .] (EstimatedCharges が次の場合) で、[Greater] (より大きい) を選択します。

10. [次よりも] で、アラームをトリガーさせる値を定義します。例えば、**200** USD などです。

    [EstimatedCharges] のメトリクス値は米ドル (USD) のみで、通貨換算は Amazon Services LLC が行います。詳細については、「AWS Billing とは」を参照してください。

    > **Note**
    >
    > しきい値を定義すると、プレビューグラフに当月の予想請求額が表示されます。

11. [追加設定] を選択し、次の操作を行います。

    - [Datapoints to alarm] (アラームを実行するデータポイント) で、1 個中 1 個 を指定します。
    - [Missing data treatment] (欠落データの処理) で、[Treat missing data as missing] (欠落データを欠落として処理) を選択します。

12. [Next] (次へ) をクリックします。

13. [通知] で、[アラーム状態] が選択されていることを確認します。次に、アラームが ALARM 状態の時に通知される Amazon SNS トピックを選択します。Amazon SNS トピックには E メールアドレスを含めることができるため、請求金額が指定したしきい値を超えたときに E メールを受信できます。

    既存の Amazon SNS トピックを選択するか、新しい Amazon SNS トピックを作成するか、またはトピック ARN を使用して他のアカウントに通知することができます。同じアラーム状態または異なるアラーム状態について複数の通知を送信する場合は、[Add notification] (通知の追加) を選択します。

14. [Next] (次へ) をクリックします。

15. [Name and description] (名前と説明) で、アラームの名前と説明を入力します。

    - (オプション) アラームの説明を入力します。

16. [Next] (次へ) をクリックします。

17. [Preview and create] (プレビューと作成) で、設定が正しいことを確認してから、[Create alarm] (アラームの作成) を選択します。

ステップ 3: アラームの状態をチェックする

先ほど作成した請求アラームの状態をチェックします。

アラームのステータスを確認するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. 必要に応じて、リージョンを米国東部 (バージニア) に変更します。請求メトリクスデータは、このリージョンに保存され、世界全体の請求額を反映します。
3. ナビゲーションペインで、[Alarms] (アラーム) を選択します。
4. ポリシーの横にあるチェックボックスをオンにします。サブスクリプションが確認されるまで、「保留中の確認」と表示されます。サブスクリプションが確認された後、更新されたステータスを表示するためにコンソールを更新します。

ステップ 4: 請求アラームを編集する

例えば、AWS で毎月消費する金額を 200 USD から 400 USD に増やしたいとします。既存の請求アラームを編集して、アラームがトリガーされるしきい値の金額を上げることができます。

請求アラームを編集するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. 必要に応じて、リージョンを米国東部 (バージニア) に変更します。請求メトリクスデータは、このリージョンに保存され、世界全体の請求額を反映します。
3. ナビゲーションペインで、[Alarms] (アラーム) を選択します。
4. アラームの横にあるチェックボックスをオンにして、[アクション]、[変更] の順に選択します。
5. [Whenever my total AWS charges for the month exceed (1 か月の AWS ご利用料金総額を超過するたび)] で、超過した場合にアラームがトリガーされ E メール通知が送信される新規の金額を指定します。
6. [Save changes] (変更の保存) をクリックします。

ステップ 5: 請求アラームを削除する

不要になった請求アラームは削除できます。

請求アラームを削除するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. 必要に応じて、リージョンを米国東部 (バージニア) に変更します。請求メトリクスデータは、このリージョンに保存され、世界全体の請求額を反映します。

3. ナビゲーションペインで、[Alarms] (アラーム) を選択します。
4. アラームの横にあるチェックボックスをオンにして、[アクション]、[削除] の順に選択します。
5. 確認を求めるメッセージが表示されたら、[Yes、Delete] を選択します。

メトリクスを CloudWatch にパブリッシュする

このシナリオでは、AWS Command Line Interface (AWS CLI) を使用して GetStarted という仮想アプリケーションの 1 つのメトリクスを発行します。まだ AWS CLI をインストールして設定していない場合は、AWS Command Line Interface ユーザーガイドの「AWS Command Line Interface のセットアップ」を参照してください。

タスク

- ステップ 1: データ構成を定義する
- ステップ 2: CloudWatch にメトリックスを追加する
- CloudWatch から統計情報を取得する
- ステップ 4: コンソールでグラフを表示する

ステップ 1: データ構成を定義する

このシナリオでは、アプリケーションのリクエストレイテンシーを追跡するデータポイントを発行します。メトリクスの名前と名前空間は、わかりやすいものを選択します。この例では、メトリクスに RequestLatency という名前を付けて、すべてのデータポイントを GetStarted という名前空間に入れます。

3 時間分のレイテンシーデータを集合的に示すデータポイントをいくつか発行します。raw データは、3 時間にわたって分散された 15 個のリクエストレイテンシーの読み取り値で構成されています。読み取り値の単位はミリ秒です。

- 1 時間目: 87、51、125、235
- 2 時間目: 121、113、189、65、89
- 3 時間目: 100、47、133、98、100、328

データを CloudWatch にパブリッシュするときは、単一のデータポイントとしてパブリッシュすることも、複数のデータポイントを集約したセット (統計セット) としてパブリッ

シュすることもできます。 メトリクス集約の最小単位は 1 分間です。データポイントを集約して統計セットとして CloudWatch にパブリッシュするときに、4 つの事前定義キー（ Sum、Minimum、Maximum、SampleCount ）を指定できます。

ここでは、1 時間目のデータポイントを個別のデータポイントとして発行します。2 時間目と 3 時間目のデータについては、時間ごとにデータポイントを集約して統計セットとして発行します。キーの値を次の表に示します。

時間	raw データ	Sum	Minimum	[Maximum] (最大)	SampleCount
1	87				
1	51				
1	125				
1	235				
2	121, 113, 189, 65, 89	577	65	189	5
3	100, 47, 133, 98, 100, 328	806	47	328	6

ステップ 2: CloudWatch にメトリックスを追加する

データ構成の定義が完了すると、データを追加できる状態になります。

CloudWatch にデータポイントを発行するには

1. コマンドプロンプトで、次の put-metric-data コマンドを実行し、最初の 1 時間のデータを追加します。サンプルのタイムスタンプを、協定世界時 (UTC) のタイプスタンプ (2 時間前) に置き換えます。

```
aws cloudwatch put-metric-data --metric-name RequestLatency --namespace GetStarted
 \
--timestamp 2016-10-14T20:30:00Z --value 87 --unit Milliseconds
aws cloudwatch put-metric-data --metric-name RequestLatency --namespace GetStarted
 \
--timestamp 2016-10-14T20:30:00Z --value 51 --unit Milliseconds
```

```
aws cloudwatch put-metric-data --metric-name RequestLatency --namespace GetStarted
 \
--timestamp 2016-10-14T20:30:00Z --value 125 --unit Milliseconds
aws cloudwatch put-metric-data --metric-name RequestLatency --namespace GetStarted
 \
--timestamp 2016-10-14T20:30:00Z --value 235 --unit Milliseconds
```

2. 2 時間目のデータを追加します。最初の 1 時間よりも 1 時間遅いタイムスタンプを使用します。

```
aws cloudwatch put-metric-data --metric-name RequestLatency --namespace GetStarted
 \
--timestamp 2016-10-14T21:30:00Z --statistic-values
 Sum=577,Minimum=65,Maximum=189,SampleCount=5 --unit Milliseconds
```

3. 3 時間目のデータを追加します。タイムスタンプは省略し、デフォルトで現在の時刻に設定されるようにします。

```
aws cloudwatch put-metric-data --metric-name RequestLatency --namespace GetStarted
 \
--statistic-values Sum=806,Minimum=47,Maximum=328,SampleCount=6 --unit Milliseconds
```

CloudWatch から統計情報を取得する

CloudWatch にメトリクスがパブリッシュされたので、次のように get-metric-statistics コマンドを使用して、そのメトリクスに基づく統計情報を取得することができます。--start-time と --end-time は、最も早く発行したタイムスタンプが含まれるように、十分な時間範囲を指定してください。

```
aws cloudwatch get-metric-statistics --namespace GetStarted --metric-name
 RequestLatency --statistics Average \
--start-time 2016-10-14T00:00:00Z --end-time 2016-10-15T00:00:00Z --period 60
```

出力例を次に示します。

```
{
 "Datapoints": [],
 "Label": "Request:Latency"
}
```

ステップ 4: コンソールでグラフを表示する

CloudWatch にメトリックスを公開したら、CloudWatch コンソールを使用して統計グラフを表示できます。

統計情報のグラフをコンソールで表示するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで [メトリクス] を選択します。
3. [すべてのメトリクス] タブで、検索ボックスに「RequestLatency」と入力し、Enter キーを押します。
4. [RequestLatency] メトリクスのチェックボックスを選択します。メトリクスのデータのグラフが上のペインに表示されます。

詳細については、「メトリクスのグラフ化」を参照してください。

AWS SDK で CloudWatch を使用する

AWS ソフトウェア開発キット (SDK) は、多くの一般的なプログラミング言語で使用できます。各 SDK には、デベロッパーが好みの言語でアプリケーションを簡単に構築できるようにする API、コード例、およびドキュメントが提供されています。

SDK ドキュメント	コードの例
AWS SDK for C++	AWS SDK for C++ コードの例
AWS CLI	AWS CLI コードの例
AWS SDK for Go	AWS SDK for Go コードの例
AWS SDK for Java	AWS SDK for Java コードの例
AWS SDK for JavaScript	AWS SDK for JavaScript コードの例
AWS SDK for Kotlin	AWS SDK for Kotlin コードの例
AWS SDK for .NET	AWS SDK for .NET コードの例
AWS SDK for PHP	AWS SDK for PHP コードの例
AWS Tools for PowerShell	Tools for PowerShell のコード例
AWS SDK for Python (Boto3)	AWS SDK for Python (Boto3) コードの例
AWS SDK for Ruby	AWS SDK for Ruby コードの例
AWS SDK for Rust	AWS SDK for Rust コードの例
AWS SDK for SAP ABAP	AWS SDK for SAP ABAP コードの例
AWS SDK for Swift	AWS SDK for Swift コードの例

CloudWatch の固有の例については、「AWS SDK を使用した CloudWatch のコードの例」を参照してください。

可用性の例

必要なものが見つからなかった場合。このページの下側にある [Provide feedback (フィードバックを送信)] リンクから、コードの例をリクエストしてください。

AWS SDK を使用した CloudWatch のコードの例

次に、AWS Software Development Kit (SDK) から CloudWatch を使用する場合のコード例を示します。

アクションはより大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。アクションは個々のサービス機能を呼び出す方法を示していますが、関連するシナリオやサービス間の例ではアクションのコンテキストが確認できます。

「シナリオ」は、同じサービス内で複数の関数を呼び出して、特定のタスクを実行する方法を示すコード例です。

クロスサービスの例は、複数の AWS のサービス で動作するサンプルアプリケーションです。

AWS SDK デベロッパーガイドとコード例の完全なリストについては、「AWS SDK で CloudWatch を使用する」を参照してください。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

開始方法

Hello CloudWatch

次のコード例は、CloudWatch の使用を開始する方法を示しています。

.NET

AWS SDK for .NET

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
using Amazon.CloudWatch;
using Amazon.CloudWatch.Model;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;
```

```
namespace CloudWatchActions;

public static class HelloCloudWatch
{
    static async Task Main(string[] args)
    {
        // Use the AWS .NET Core Setup package to set up dependency injection for
 the Amazon CloudWatch service.
        // Use your AWS profile name, or leave it blank to use the default
 profile.
        using var host = Host.CreateDefaultBuilder(args)
            .ConfigureServices((_, services) =>
                services.AddAWSService<IAmazonCloudWatch>()
            ).Build();

        // Now the client is available for injection.
        var cloudWatchClient =
 host.Services.GetRequiredService<IAmazonCloudWatch>();

        // You can use await and any of the async methods to get a response.
        var metricNamespace = "AWS/Billing";
        var response = await cloudWatchClient.ListMetricsAsync(new
 ListMetricsRequest
        {
            Namespace = metricNamespace
        });
        Console.WriteLine($"Hello Amazon CloudWatch! Following are some metrics
 available in the {metricNamespace} namespace:");
        Console.WriteLine();
        foreach (var metric in response.Metrics.Take(5))
        {
            Console.WriteLine($"\tMetric: {metric.MetricName}");
            Console.WriteLine($"\tNamespace: {metric.Namespace}");
            Console.WriteLine($"\tDimensions: {string.Join(", ",
 metric.Dimensions.Select(m => $"{m.Name}:{m.Value}"))}");
            Console.WriteLine();
        }
    }
}
```

- API の詳細については、「AWS SDK for .NET API リファレンス」の「ListMetrics」を参照してください。

Java

SDK for Java 2.x

> **Note**
>
> GitHub には、その他のリソースもあります。完全な例を見つけて、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.cloudwatch.CloudWatchClient;
import software.amazon.awssdk.services.cloudwatch.model.CloudWatchException;
import software.amazon.awssdk.services.cloudwatch.model.ListMetricsRequest;
import software.amazon.awssdk.services.cloudwatch.paginators.ListMetricsIterable;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class HelloService {
    public static void main(String[] args) {
        final String usage = """

                Usage:
                  <namespace>\s

                Where:
                  namespace - The namespace to filter against (for example, AWS/
EC2).\s
                """;

        if (args.length != 1) {
            System.out.println(usage);
            System.exit(1);
        }
```

```
        String namespace = args[0];
        Region region = Region.US_EAST_1;
        CloudWatchClient cw = CloudWatchClient.builder()
                .region(region)
                .build();

        listMets(cw, namespace);
        cw.close();
    }

    public static void listMets(CloudWatchClient cw, String namespace) {
        try {
            ListMetricsRequest request = ListMetricsRequest.builder()
                    .namespace(namespace)
                    .build();

            ListMetricsIterable listRes = cw.listMetricsPaginator(request);
            listRes.stream()
                    .flatMap(r -> r.metrics().stream())
                    .forEach(metrics -> System.out.println(" Retrieved metric is:
 " + metrics.metricName()));

        } catch (CloudWatchException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }
}
```

- API の詳細については、「AWS SDK for Java 2.x API リファレンス」の「ListMetrics」を参照してください。

Kotlin

SDK for Kotlin

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
/**
Before running this Kotlin code example, set up your development environment,
including your credentials.

For more information, see the following documentation topic:
https://docs.aws.amazon.com/sdk-for-kotlin/latest/developer-guide/setup.html
 */
suspend fun main(args: Array<String>) {
    val usage = """
        Usage:
           <namespace>
        Where:
           namespace - The namespace to filter against (for example, AWS/EC2).
    """

    if (args.size != 1) {
        println(usage)
        exitProcess(0)
    }

    val namespace = args[0]
    listAllMets(namespace)
}

suspend fun listAllMets(namespaceVal: String?) {
    val request = ListMetricsRequest {
        namespace = namespaceVal
    }

    CloudWatchClient { region = "us-east-1" }.use { cwClient ->
        cwClient.listMetricsPaginated(request)
            .transform { it.metrics?.forEach { obj -> emit(obj) } }
            .collect { obj ->
                println("Name is ${obj.metricName}")
                println("Namespace is ${obj.namespace}")
            }
    }
}
```

- API の詳細については、「AWS SDK for Kotlin API リファレンス」の「ListMetrics」を参照してください。

コードの例

- AWS SDK を使用した CloudWatch のアクション
  - AWS SDK または CLI で DeleteAlarms を使用する
  - AWS SDK または CLI で DeleteAnomalyDetector を使用する
  - AWS SDK または CLI で DeleteDashboards を使用する
  - AWS SDK または CLI で DescribeAlarmHistory を使用する
  - AWS SDK または CLI で DescribeAlarms を使用する
  - AWS SDK または CLI で DescribeAlarmsForMetric を使用する
  - AWS SDK または CLI で DescribeAnomalyDetectors を使用する
  - AWS SDK または CLI で DisableAlarmActions を使用する
  - AWS SDK または CLI で EnableAlarmActions を使用する
  - AWS SDK または CLI で GetDashboard を使用する
  - AWS SDK または CLI で GetMetricData を使用する
  - AWS SDK または CLI で GetMetricStatistics を使用する
  - AWS SDK または CLI で GetMetricWidgetImage を使用する
  - AWS SDK または CLI で ListDashboards を使用する
  - AWS SDK または CLI で ListMetrics を使用する
  - AWS SDK または CLI で PutAnomalyDetector を使用する
  - AWS SDK または CLI で PutDashboard を使用する
  - AWS SDK または CLI で PutMetricAlarm を使用する
  - AWS SDK または CLI で PutMetricData を使用する
- AWS SDK を使用した CloudWatch のシナリオ
  - AWS SDK を使用した CloudWatch アラームの開始方法
  - AWS SDK を使用して CloudWatch のメトリクス、ダッシュボード、およびアラームの使用を開始する
  - AWS SDK により CloudWatch メトリクスとアラームを管理する
- AWS SDK を使用した CloudWatch のクロスサービスの例
  - AWS SDK を使用した Amazon DynamoDB のパフォーマンスのモニタリング

AWS SDK を使用した CloudWatch のアクション

次のコード例は、AWS SDKを使用して個々の CloudWatch アクションを実行する方法を示しています。これらは CloudWatch API を呼び出すもので、コンテキスト内で実行する必要がある大規模なプログラムからのコード抜粋です。それぞれの例には、GitHub へのリンクがあり、そこにはコードの設定と実行に関する説明が記載されています。

以下の例には、最も一般的に使用されるアクションのみ含まれています。詳細な一覧については、「Amazon CloudWatch API Reference」(Amazon CloudWatch API リファレンス) を参照してください。

例

- AWS SDK または CLI で DeleteAlarms を使用する
- AWS SDK または CLI で DeleteAnomalyDetector を使用する
- AWS SDK または CLI で DeleteDashboards を使用する
- AWS SDK または CLI で DescribeAlarmHistory を使用する
- AWS SDK または CLI で DescribeAlarms を使用する
- AWS SDK または CLI で DescribeAlarmsForMetric を使用する
- AWS SDK または CLI で DescribeAnomalyDetectors を使用する
- AWS SDK または CLI で DisableAlarmActions を使用する
- AWS SDK または CLI で EnableAlarmActions を使用する
- AWS SDK または CLI で GetDashboard を使用する
- AWS SDK または CLI で GetMetricData を使用する
- AWS SDK または CLI で GetMetricStatistics を使用する
- AWS SDK または CLI で GetMetricWidgetImage を使用する
- AWS SDK または CLI で ListDashboards を使用する
- AWS SDK または CLI で ListMetrics を使用する
- AWS SDK または CLI で PutAnomalyDetector を使用する
- AWS SDK または CLI で PutDashboard を使用する
- AWS SDK または CLI で PutMetricAlarm を使用する
- AWS SDK または CLI で PutMetricData を使用する

AWS SDK または CLI で **DeleteAlarms** を使用する

以下のコード例は、DeleteAlarms の使用方法を示しています。

アクション例は、より大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。次のコード例で、このアクションのコンテキストを確認できます。

- アラームの使用を開始
- メトリクス、ダッシュボード、およびアラームの使用を開始する
- メトリクスとアラームを管理する

.NET

AWS SDK for .NET

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
/// <summary>
/// Delete a list of alarms from CloudWatch.
/// </summary>
/// <param name="alarmNames">A list of names of alarms to delete.</param>
/// <returns>True if successful.</returns>
public async Task<bool> DeleteAlarms(List<string> alarmNames)
{
    var deleteAlarmsResult = await _amazonCloudWatch.DeleteAlarmsAsync(
        new DeleteAlarmsRequest()
        {
            AlarmNames = alarmNames
        });

    return deleteAlarmsResult.HttpStatusCode == HttpStatusCode.OK;
}
```

- API の詳細については、「AWS SDK for .NET API リファレンス」の「DeleteAlarms」を参照してください。

C++

SDK for C++

> **Note**
>
> GitHub には、その他のリソースもあります。完全な例を見つけて、AWS コード例リポジトリでの設定と実行の方法を確認してください。

必要なファイルを含めます。

```
#include <aws/core/Aws.h>
#include <aws/monitoring/CloudWatchClient.h>
#include <aws/monitoring/model/DeleteAlarmsRequest.h>
#include <iostream>
```

アラームを削除します。

```
        Aws::CloudWatch::CloudWatchClient cw;
        Aws::CloudWatch::Model::DeleteAlarmsRequest request;
        request.AddAlarmNames(alarm_name);

        auto outcome = cw.DeleteAlarms(request);
        if (!outcome.IsSuccess())
        {
            std::cout << "Failed to delete CloudWatch alarm:" <<
                outcome.GetError().GetMessage() << std::endl;
        }
        else
        {
            std::cout << "Successfully deleted CloudWatch alarm " << alarm_name
                << std::endl;
        }
```

- API の詳細については、「AWS SDK for C++ API リファレンス」の「DeleteAlarms」を参照してください。

CLI

AWS CLI

アラームを削除するには

次の例は、delete-alarms コマンドを使用して、「myalarm」という名前の Amazon CloudWatch アラームを削除します。

```
aws cloudwatch delete-alarms --alarm-names myalarm
```

出力:

```
This command returns to the prompt if successful.
```

- API の詳細については、AWS CLI コマンドリファレンスの「DeleteAlarms」を参照してください。

Java

SDK for Java 2.x

> **Note**
>
> GitHub には、その他のリソースもあります。完全な例を見つけて、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.cloudwatch.CloudWatchClient;
import software.amazon.awssdk.services.cloudwatch.model.CloudWatchException;
import software.amazon.awssdk.services.cloudwatch.model.DeleteAlarmsRequest;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
```

```
 */

public class DeleteAlarm {
    public static void main(String[] args) {
        final String usage = """

                Usage:
                  <alarmName>

                Where:
                  alarmName - An alarm name to delete (for example, MyAlarm).
                """;

        if (args.length != 1) {
            System.out.println(usage);
            System.exit(1);
        }

        String alarmName = args[0];
        Region region = Region.US_EAST_2;
        CloudWatchClient cw = CloudWatchClient.builder()
                .region(region)
                .build();

        deleteCWAlarm(cw, alarmName);
        cw.close();
    }

    public static void deleteCWAlarm(CloudWatchClient cw, String alarmName) {
        try {
            DeleteAlarmsRequest request = DeleteAlarmsRequest.builder()
                    .alarmNames(alarmName)
                    .build();

            cw.deleteAlarms(request);
            System.out.printf("Successfully deleted alarm %s", alarmName);

        } catch (CloudWatchException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }
}
```

- API の詳細については、「AWS SDK for Java 2.x API リファレンス」の「DeleteAlarms」を参照してください。

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

SDK モジュールとクライアントモジュールをインポートし、API を呼び出します。

```
import { DeleteAlarmsCommand } from "@aws-sdk/client-cloudwatch";
import { client } from "../libs/client.js";

const run = async () => {
  const command = new DeleteAlarmsCommand({
    AlarmNames: [process.env.CLOUDWATCH_ALARM_NAME], // Set the value of
 CLOUDWATCH_ALARM_NAME to the name of an existing alarm.
  });

  try {
    return await client.send(command);
  } catch (err) {
    console.error(err);
  }
};

export default run();
```

別のモジュールでクライアントを作成し、エクスポートします。

```
import { CloudWatchClient } from "@aws-sdk/client-cloudwatch";
```

```
export const client = new CloudWatchClient({});
```

- 詳細については、「AWS SDK for JavaScript デベロッパーガイド」を参照してください。
- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「DeleteAlarms」を参照してください。

SDK for JavaScript (v2)

> **Note**
>
> GitHub には、その他のリソースもあります。完全な例を見つけて、AWS コード例リポジトリでの設定と実行の方法を確認してください。

SDK モジュールとクライアントモジュールをインポートし、API を呼び出します。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create CloudWatch service object
var cw = new AWS.CloudWatch({ apiVersion: "2010-08-01" });

var params = {
  AlarmNames: ["Web_Server_CPU_Utilization"],
};

cw.deleteAlarms(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- 詳細については、AWS SDK for JavaScript デベロッパーガイドを参照してください。
- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「DeleteAlarms」を参照してください。

Kotlin

SDK for Kotlin

> **Note**
>
> GitHub には、その他のリソースもあります。完全な例を見つけて、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
suspend fun deleteAlarm(alarmNameVal: String) {
    val request = DeleteAlarmsRequest {
        alarmNames = listOf(alarmNameVal)
    }

    CloudWatchClient { region = "us-east-1" }.use { cwClient ->
        cwClient.deleteAlarms(request)
        println("Successfully deleted alarm $alarmNameVal")
    }
}
```

- API の詳細については、AWS SDK for Kotlin API リファレンスの「DeleteAlarms」を参照してください。

Python

SDK for Python (Boto3)

> **Note**
>
> GitHub には、その他のリソースもあります。完全な例を見つけて、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
class CloudWatchWrapper:
    """Encapsulates Amazon CloudWatch functions."""

    def __init__(self, cloudwatch_resource):
```

```
        """
        :param cloudwatch_resource: A Boto3 CloudWatch resource.
        """
        self.cloudwatch_resource = cloudwatch_resource


    def delete_metric_alarms(self, metric_namespace, metric_name):
        """
        Deletes all of the alarms that are currently watching the specified
metric.

        :param metric_namespace: The namespace of the metric.
        :param metric_name: The name of the metric.
        """
        try:
            metric = self.cloudwatch_resource.Metric(metric_namespace,
metric_name)
            metric.alarms.delete()
            logger.info(
                "Deleted alarms for metric %s.%s.", metric_namespace, metric_name
            )
        except ClientError:
            logger.exception(
                "Couldn't delete alarms for metric %s.%s.",
                metric_namespace,
                metric_name,
            )
            raise
```

- API の詳細については、AWS SDK for Python (Boto3) API リファレンスの「DeleteAlarms」を参照してください。

SAP ABAP

SDK for SAP ABAP

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
TRY.
    lo_cwt->deletealarms(
      it_alarmnames = it_alarm_names
    ).
    MESSAGE 'Alarms deleted.' TYPE 'I'.
  CATCH /aws1/cx_cwtresourcenotfound .
    MESSAGE 'Resource being accessed is not found.' TYPE 'E'.
ENDTRY.
```

- API の詳細については、「AWS SDK for SAP ABAP API リファレンス」の「DeleteAlarms」を参照してください。

AWS SDK デベロッパーガイドとコード例の完全なリストについては、「AWS SDK で CloudWatch を使用する」を参照してください。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK または CLI で **DeleteAnomalyDetector** を使用する

以下のコード例は、DeleteAnomalyDetector の使用方法を示しています。

アクション例は、より大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。次のコード例で、このアクションのコンテキストを確認できます。

- メトリクス、ダッシュボード、およびアラームの使用を開始する

.NET

AWS SDK for .NET

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Delete a single metric anomaly detector.
    /// </summary>
    /// <param name="anomalyDetector">The anomaly detector to delete.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DeleteAnomalyDetector(SingleMetricAnomalyDetector
anomalyDetector)
    {
        var deleteAnomalyDetectorResponse = await
_amazonCloudWatch.DeleteAnomalyDetectorAsync(
            new DeleteAnomalyDetectorRequest()
            {
                SingleMetricAnomalyDetector = anomalyDetector
            });

        return deleteAnomalyDetectorResponse.HttpStatusCode == HttpStatusCode.OK;
    }
```

- API の詳細については、「AWS SDK for .NET API Reference」(API リファレンス) の「DeleteAnomalyDetector」を参照してください。

Java

SDK for Java 2.x

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
   public static void deleteAnomalyDetector(CloudWatchClient cw, String
fileName) {
       try {
           // Read values from the JSON file.
           JsonParser parser = new JsonFactory().createParser(new
File(fileName));
           com.fasterxml.jackson.databind.JsonNode rootNode = new
ObjectMapper().readTree(parser);
           String customMetricNamespace =
rootNode.findValue("customMetricNamespace").asText();
           String customMetricName =
rootNode.findValue("customMetricName").asText();

           SingleMetricAnomalyDetector singleMetricAnomalyDetector =
SingleMetricAnomalyDetector.builder()
                   .metricName(customMetricName)
                   .namespace(customMetricNamespace)
                   .stat("Maximum")
                   .build();

           DeleteAnomalyDetectorRequest request =
DeleteAnomalyDetectorRequest.builder()
                   .singleMetricAnomalyDetector(singleMetricAnomalyDetector)
                   .build();

           cw.deleteAnomalyDetector(request);
           System.out.println("Successfully deleted the Anomaly Detector.");

       } catch (CloudWatchException e) {
           System.err.println(e.awsErrorDetails().errorMessage());
           System.exit(1);
       } catch (IOException e) {
           e.printStackTrace();
       }
   }
```

- API の詳細については、「AWS SDK for Java 2.x API Reference」(API リファレンス) の「DeleteAnomalyDetector」を参照してください。

Kotlin

SDK for Kotlin

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
suspend fun deleteAnomalyDetector(fileName: String) {
    // Read values from the JSON file.
    val parser = JsonFactory().createParser(File(fileName))
    val rootNode = ObjectMapper().readTree<JsonNode>(parser)
    val customMetricNamespace =
 rootNode.findValue("customMetricNamespace").asText()
    val customMetricName = rootNode.findValue("customMetricName").asText()

    val singleMetricAnomalyDetectorVal = SingleMetricAnomalyDetector {
        metricName = customMetricName
        namespace = customMetricNamespace
        stat = "Maximum"
    }

    val request = DeleteAnomalyDetectorRequest {
        singleMetricAnomalyDetector = singleMetricAnomalyDetectorVal
    }

    CloudWatchClient { region = "us-east-1" }.use { cwClient ->
        cwClient.deleteAnomalyDetector(request)
        println("Successfully deleted the Anomaly Detector.")
    }
}
```

- API の詳細については、「AWS SDK for Kotlin API リファレンス」の「DeleteAnomalyDetector」を参照してください。

AWS SDK デベロッパーガイドとコード例の完全なリストについては、「AWS SDK で CloudWatch を使用する」を参照してください。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK または CLI で **DeleteDashboards** を使用する

以下のコード例は、DeleteDashboards の使用方法を示しています。

アクション例は、より大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。次のコード例で、このアクションのコンテキストを確認できます。

- メトリクス、ダッシュボード、およびアラームの使用を開始する

.NET

AWS SDK for .NET

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Delete a list of CloudWatch dashboards.
    /// </summary>
    /// <param name="dashboardNames">List of dashboard names to delete.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DeleteDashboards(List<string> dashboardNames)
    {
        var deleteDashboardsResponse = await
_amazonCloudWatch.DeleteDashboardsAsync(
            new DeleteDashboardsRequest()
            {
                DashboardNames = dashboardNames
            });

        return deleteDashboardsResponse.HttpStatusCode == HttpStatusCode.OK;
    }
```

- API の詳細については、「AWS SDK for .NET API リファレンス」の「DeleteDashboards」を参照してください。

Java

SDK for Java 2.x

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
   public static void deleteDashboard(CloudWatchClient cw, String dashboardName)
{
       try {
           DeleteDashboardsRequest dashboardsRequest =
DeleteDashboardsRequest.builder()
                   .dashboardNames(dashboardName)
                   .build();
           cw.deleteDashboards(dashboardsRequest);
           System.out.println(dashboardName + " was successfully deleted.");

       } catch (CloudWatchException e) {
           System.err.println(e.getMessage());
           System.exit(1);
       }
   }
```

- API の詳細については、「AWS SDK for Java 2.x API リファレンス」の「DeleteDashboards」を参照してください。

Kotlin

SDK for Kotlin

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
suspend fun deleteDashboard(dashboardName: String) {
    val dashboardsRequest = DeleteDashboardsRequest {
        dashboardNames = listOf(dashboardName)
    }
    CloudWatchClient { region = "us-east-1" }.use { cwClient ->
        cwClient.deleteDashboards(dashboardsRequest)
        println("$dashboardName was successfully deleted.")
    }
}
```

- API の詳細については、「AWS SDK for Kotlin API リファレンス」の「DeleteDashboards」を参照してください。

PowerShell

Tools for PowerShell

例 1: 指定したダッシュボードを削除し、次に進む前に確認を促します。確認を省略するには、-Force スイッチをコマンドに追加します。

```
Remove-CWDashboard -DashboardName Dashboard1
```

- API の詳細については、「AWS Tools for PowerShell コマンドレットリファレンス」の「DeleteDashboards」を参照してください。

AWS SDK デベロッパーガイドとコード例の完全なリストについては、「AWS SDK で CloudWatch を使用する」を参照してください。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK または CLI で **DescribeAlarmHistory** を使用する

以下のコード例は、DescribeAlarmHistory の使用方法を示しています。

アクション例は、より大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。次のコード例で、このアクションのコンテキストを確認できます。

- メトリクス、ダッシュボード、およびアラームの使用を開始する

.NET

AWS SDK for .NET

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Describe the history of an alarm for a number of days in the past.
    /// </summary>
    /// <param name="alarmName">The name of the alarm.</param>
    /// <param name="historyDays">The number of days in the past.</param>
    /// <returns>The list of alarm history data.</returns>
    public async Task<List<AlarmHistoryItem>> DescribeAlarmHistory(string
alarmName, int historyDays)
    {
        List<AlarmHistoryItem> alarmHistory = new List<AlarmHistoryItem>();
        var paginatedAlarmHistory =
_amazonCloudWatch.Paginators.DescribeAlarmHistory(
            new DescribeAlarmHistoryRequest()
            {
                AlarmName = alarmName,
                EndDateUtc = DateTime.UtcNow,
                HistoryItemType = HistoryItemType.StateUpdate,
                StartDateUtc = DateTime.UtcNow.AddDays(-historyDays)
            });

        await foreach (var data in paginatedAlarmHistory.AlarmHistoryItems)
        {
            alarmHistory.Add(data);
        }
        return alarmHistory;
    }
```

- API の詳細については、「AWS SDK for .NET API リファレンス」の「DescribeAlarmHistory」を参照してください。

CLI

AWS CLI

アラームの履歴を取得するには

次の例は、describe-alarm-history コマンドを使用して、「myalarm」という名前の Amazon CloudWatch アラームの履歴を取得します。

```
aws cloudwatch describe-alarm-history --alarm-name "myalarm" --history-item-type
 StateUpdate
```

出力:

```
{
    "AlarmHistoryItems": [
        {
            "Timestamp": "2014-04-09T18:59:06.442Z",
            "HistoryItemType": "StateUpdate",
            "AlarmName": "myalarm",
            "HistoryData": "{\"version\":\"1.0\",\"oldState\":{\"stateValue
\":\"ALARM\",\"stateReason\":\"testing purposes\"},\"newState\":{\"stateValue
\":\"OK\",\"stateReason\":\"Threshold Crossed: 2 datapoints were not
 greater than the threshold (70.0). The most recent datapoints: [38.958,
 40.292].\",\"stateReasonData\":{\"version\":\"1.0\",\"queryDate\":
\"2014-04-09T18:59:06.419+0000\",\"startDate\":\"2014-04-09T18:44:00.000+0000\",
\"statistic\":\"Average\",\"period\":300,\"recentDatapoints\":[38.958,40.292],
\"threshold\":70.0}}}",
            "HistorySummary": "Alarm updated from ALARM to OK"
        },
        {
            "Timestamp": "2014-04-09T18:59:05.805Z",
            "HistoryItemType": "StateUpdate",
            "AlarmName": "myalarm",
            "HistoryData": "{\"version\":\"1.0\",\"oldState\":{\"stateValue
\":\"OK\",\"stateReason\":\"Threshold Crossed: 2 datapoints were
 not greater than the threshold (70.0). The most recent datapoints:
 [38.839999999999996, 39.714].\",\"stateReasonData\":{\"version\":
\"1.0\",\"queryDate\":\"2014-03-11T22:45:41.569+0000\",\"startDate\":
\"2014-03-11T22:30:00.000+0000\",\"statistic\":\"Average\",\"period\":300,
\"recentDatapoints\":[38.839999999999996,39.714],\"threshold\":70.0}},\"newState
\":{\"stateValue\":\"ALARM\",\"stateReason\":\"testing purposes\"}}",
            "HistorySummary": "Alarm updated from OK to ALARM"
```

```
        }
    ]
}
```

- API の詳細については、AWS CLI コマンドリファレンスの「DescribeAlarmHistory」を参照してください。

Java

SDK for Java 2.x

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
   public static void getAlarmHistory(CloudWatchClient cw, String fileName,
String date) {
       try {
           // Read values from the JSON file.
           JsonParser parser = new JsonFactory().createParser(new
File(fileName));
           com.fasterxml.jackson.databind.JsonNode rootNode = new
ObjectMapper().readTree(parser);
           String alarmName = rootNode.findValue("exampleAlarmName").asText();

           Instant start = Instant.parse(date);
           Instant endDate = Instant.now();
           DescribeAlarmHistoryRequest historyRequest =
DescribeAlarmHistoryRequest.builder()
                   .startDate(start)
                   .endDate(endDate)
                   .alarmName(alarmName)
                   .historyItemType(HistoryItemType.ACTION)
                   .build();

           DescribeAlarmHistoryResponse response =
cw.describeAlarmHistory(historyRequest);
           List<AlarmHistoryItem> historyItems = response.alarmHistoryItems();
           if (historyItems.isEmpty()) {
```

```
                System.out.println("No alarm history data found for " + alarmName
 + ".");
            } else {
                for (AlarmHistoryItem item : historyItems) {
                    System.out.println("History summary: " +
 item.historySummary());
                    System.out.println("Time stamp: " + item.timestamp());
                }
            }

        } catch (CloudWatchException | IOException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }
```

- API の詳細については、「AWS SDK for Java 2.x API リファレンス」の「DescribeAlarmHistory」を参照してください。

Kotlin

SDK for Kotlin

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
suspend fun getAlarmHistory(fileName: String, date: String) {
    // Read values from the JSON file.
    val parser = JsonFactory().createParser(File(fileName))
    val rootNode = ObjectMapper().readTree<JsonNode>(parser)
    val alarmNameVal = rootNode.findValue("exampleAlarmName").asText()
    val start = Instant.parse(date)
    val endDateVal = Instant.now()

    val historyRequest = DescribeAlarmHistoryRequest {
        startDate = aws.smithy.kotlin.runtime.time.Instant(start)
        endDate = aws.smithy.kotlin.runtime.time.Instant(endDateVal)
```

```
        alarmName = alarmNameVal
        historyItemType = HistoryItemType.Action
    }

    CloudWatchClient { credentialsProvider = EnvironmentCredentialsProvider();
 region = "us-east-1" }.use { cwClient ->
        val response = cwClient.describeAlarmHistory(historyRequest)
        val historyItems = response.alarmHistoryItems
        if (historyItems != null) {
            if (historyItems.isEmpty()) {
                println("No alarm history data found for $alarmNameVal.")
            } else {
                for (item in historyItems) {
                    println("History summary ${item.historySummary}")
                    println("Time stamp: ${item.timestamp}")
                }
            }
        }
    }
}
```

- API の詳細については、「AWS SDK for Kotlin API reference」(SDK for Kotlin API リファレンス) の「DescribeAlarmHistory」を参照してください。

AWS SDK デベロッパーガイドとコード例の完全なリストについては、「AWS SDK で CloudWatch を使用する」を参照してください。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK または CLI で **DescribeAlarms** を使用する

以下のコード例は、DescribeAlarms の使用方法を示しています。

アクション例は、より大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。次のコード例で、このアクションのコンテキストを確認できます。

- アラームの使用を開始
- メトリクス、ダッシュボード、およびアラームの使用を開始する

.NET

AWS SDK for .NET

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Describe the current alarms, optionally filtered by state.
    /// </summary>
    /// <param name="stateValue">Optional filter for alarm state.</param>
    /// <returns>The list of alarm data.</returns>
    public async Task<List<MetricAlarm>> DescribeAlarms(StateValue? stateValue =
null)
    {
        List<MetricAlarm> alarms = new List<MetricAlarm>();
        var paginatedDescribeAlarms =
_amazonCloudWatch.Paginators.DescribeAlarms(
            new DescribeAlarmsRequest()
            {
                StateValue = stateValue
            });

        await foreach (var data in paginatedDescribeAlarms.MetricAlarms)
        {
            alarms.Add(data);
        }
        return alarms;
    }
```

- API の詳細については、「AWS SDK for .NET API リファレンス」の「DescribeAlarms」を参照してください。

CLI

AWS CLI

アラームに関する情報を一覧表示するには

次の例は、describe-alarms コマンドを使用して、「myalarm」という名前のアラームに関する情報を表示します。

```
aws cloudwatch describe-alarms --alarm-names "myalarm"
```

出力:

```
{
    "MetricAlarms": [
        {
            "EvaluationPeriods": 2,
            "AlarmArn": "arn:aws:cloudwatch:us-
east-1:123456789012:alarm:myalarm",
            "StateUpdatedTimestamp": "2014-04-09T18:59:06.442Z",
            "AlarmConfigurationUpdatedTimestamp": "2012-12-27T00:49:54.032Z",
            "ComparisonOperator": "GreaterThanThreshold",
            "AlarmActions": [
                "arn:aws:sns:us-east-1:123456789012:myHighCpuAlarm"
            ],
            "Namespace": "AWS/EC2",
            "AlarmDescription": "CPU usage exceeds 70 percent",
            "StateReasonData": "{\"version\":\"1.0\",\"queryDate\":
\"2014-04-09T18:59:06.419+0000\",\"startDate\":\"2014-04-09T18:44:00.000+0000\",
\"statistic\":\"Average\",\"period\":300,\"recentDatapoints\":[38.958,40.292],
\"threshold\":70.0}",
            "Period": 300,
            "StateValue": "OK",
            "Threshold": 70.0,
            "AlarmName": "myalarm",
            "Dimensions": [
                {
                    "Name": "InstanceId",
                    "Value": "i-0c986c72"
                }
            ],
            "Statistic": "Average",
```

```
            "StateReason": "Threshold Crossed: 2 datapoints were not greater than
 the threshold (70.0). The most recent datapoints: [38.958, 40.292].",
            "InsufficientDataActions": [],
            "OKActions": [],
            "ActionsEnabled": true,
            "MetricName": "CPUUtilization"
        }
    ]
}
```

- API の詳細については、AWS CLI コマンドリファレンスの「DescribeAlarms」を参照してください。

Java

SDK for Java 2.x

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
    public static void describeAlarms(CloudWatchClient cw) {
        try {
            List<AlarmType> typeList = new ArrayList<>();
            typeList.add(AlarmType.METRIC_ALARM);

            DescribeAlarmsRequest alarmsRequest = DescribeAlarmsRequest.builder()
                    .alarmTypes(typeList)
                    .maxRecords(10)
                    .build();

            DescribeAlarmsResponse response = cw.describeAlarms(alarmsRequest);
            List<MetricAlarm> alarmList = response.metricAlarms();
            for (MetricAlarm alarm : alarmList) {
                System.out.println("Alarm name: " + alarm.alarmName());
                System.out.println("Alarm description: " +
 alarm.alarmDescription());
            }
        } catch (CloudWatchException e) {
```

```
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }
```

- API の詳細については、「AWS SDK for Java 2.x API リファレンス」の「DescribeAlarms」を参照してください。

Kotlin

SDK for Kotlin

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
suspend fun describeAlarms() {
    val typeList = ArrayList<AlarmType>()
    typeList.add(AlarmType.MetricAlarm)
    val alarmsRequest = DescribeAlarmsRequest {
        alarmTypes = typeList
        maxRecords = 10
    }

    CloudWatchClient { region = "us-east-1" }.use { cwClient ->
        val response = cwClient.describeAlarms(alarmsRequest)
        response.metricAlarms?.forEach { alarm ->
            println("Alarm name: ${alarm.alarmName}")
            println("Alarm description: ${alarm.alarmDescription}")
        }
    }
}
```

- API の詳細については、「AWS SDK for Kotlin API リファレンス」の「DescribeAlarms」を参照してください。

Ruby

SDK for Ruby

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
require "aws-sdk-cloudwatch"

# Lists the names of available Amazon CloudWatch alarms.
#
# @param cloudwatch_client [Aws::CloudWatch::Client]
#   An initialized CloudWatch client.
# @example
#   list_alarms(Aws::CloudWatch::Client.new(region: 'us-east-1'))
def list_alarms(cloudwatch_client)
  response = cloudwatch_client.describe_alarms
  if response.metric_alarms.count.positive?
    response.metric_alarms.each do |alarm|
      puts alarm.alarm_name
    end
  else
    puts "No alarms found."
  end
rescue StandardError => e
  puts "Error getting information about alarms: #{e.message}"
end
```

- API の詳細については、「AWS SDK for Ruby API リファレンス」の「DescribeAlarms」を参照してください。

SAP ABAP

SDK for SAP ABAP

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
    TRY.
        oo_result = lo_cwt->describealarms(                 " oo_result is
 returned for testing purposes. "
          it_alarmnames = it_alarm_names
        ).
        MESSAGE 'Alarms retrieved.' TYPE 'I'.
      CATCH /aws1/cx_rt_service_generic INTO DATA(lo_exception).
        DATA(lv_error) = |"{ lo_exception->av_err_code }" - { lo_exception-
>av_err_msg }|.
        MESSAGE lv_error TYPE 'E'.
    ENDTRY.
```

- API の詳細については、「AWS SDK for SAP ABAP API リファレンス」の「DescribeAlarms」を参照してください。

AWS SDK デベロッパーガイドとコード例の完全なリストについては、「AWS SDK で CloudWatch を使用する」を参照してください。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK または CLI で **DescribeAlarmsForMetric** を使用する

以下のコード例は、DescribeAlarmsForMetric の使用方法を示しています。

アクション例は、より大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。次のコード例で、このアクションのコンテキストを確認できます。

- メトリクス、ダッシュボード、およびアラームの使用を開始する
- メトリクスとアラームを管理する

.NET

AWS SDK for .NET

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Describe the current alarms for a specific metric.
    /// </summary>
    /// <param name="metricNamespace">The namespace of the metric.</param>
    /// <param name="metricName">The name of the metric.</param>
    /// <returns>The list of alarm data.</returns>
    public async Task<List<MetricAlarm>> DescribeAlarmsForMetric(string
metricNamespace, string metricName)
    {
        var alarmsResult = await _amazonCloudWatch.DescribeAlarmsForMetricAsync(
            new DescribeAlarmsForMetricRequest()
            {
                Namespace = metricNamespace,
                MetricName = metricName
            });

        return alarmsResult.MetricAlarms;
    }
```

- API の詳細については、「AWS SDK for .NET API リファレンス」の「DescribeAlarmsForMetric」を参照してください。

C++

SDK for C++

> **Note**
>
> GitHub には、その他のリソースもあります。完全な例を見つけて、AWS コード例リポジトリでの設定と実行の方法を確認してください。

必要なファイルを含めます。

```
#include <aws/core/Aws.h>
#include <aws/monitoring/CloudWatchClient.h>
#include <aws/monitoring/model/DescribeAlarmsRequest.h>
#include <aws/monitoring/model/DescribeAlarmsResult.h>
#include <iomanip>
#include <iostream>
```

mon-describe-alarms

```
        Aws::CloudWatch::CloudWatchClient cw;
        Aws::CloudWatch::Model::DescribeAlarmsRequest request;
        request.SetMaxRecords(1);

        bool done = false;
        bool header = false;
        while (!done)
        {
            auto outcome = cw.DescribeAlarms(request);
            if (!outcome.IsSuccess())
            {
                std::cout << "Failed to describe CloudWatch alarms:" <<
                    outcome.GetError().GetMessage() << std::endl;
                break;
            }

            if (!header)
            {
                std::cout << std::left <<
                    std::setw(32) << "Name" <<
```

```
                    std::setw(64) << "Arn" <<
                    std::setw(64) << "Description" <<
                    std::setw(20) << "LastUpdated" <<
                    std::endl;
                header = true;
            }

            const auto &alarms = outcome.GetResult().GetMetricAlarms();
            for (const auto &alarm : alarms)
            {
                std::cout << std::left <<
                    std::setw(32) << alarm.GetAlarmName() <<
                    std::setw(64) << alarm.GetAlarmArn() <<
                    std::setw(64) << alarm.GetAlarmDescription() <<
                    std::setw(20) <<
                    alarm.GetAlarmConfigurationUpdatedTimestamp().ToGmtString(
                        SIMPLE_DATE_FORMAT_STR) <<
                    std::endl;
            }

            const auto &next_token = outcome.GetResult().GetNextToken();
            request.SetNextToken(next_token);
            done = next_token.empty();
        }
```

- API の詳細については、「AWS SDK for C++ API リファレンス」の「DescribeAlarmsForMetric」を参照してください。

CLI

AWS CLI

メトリクスに関連するアラームについての情報を表示するには

次の例は、describe-alarms-for-metric コマンドを使用して、Amazon EC2 CPUUtilization メトリクスと、ID i-0c986c72 のインスタンスに関連するすべてのアラームに関する情報を表示します。

```
aws cloudwatch describe-alarms-for-metric --metric-name CPUUtilization --
namespace AWS/EC2 --dimensions Name=InstanceId,Value=i-0c986c72
```

出力:


```
{
    "MetricAlarms": [
        {
            "EvaluationPeriods": 10,
            "AlarmArn": "arn:aws:cloudwatch:us-
east-1:111122223333:alarm:myHighCpuAlarm2",
            "StateUpdatedTimestamp": "2013-10-30T03:03:51.479Z",
            "AlarmConfigurationUpdatedTimestamp": "2013-10-30T03:03:50.865Z",
            "ComparisonOperator": "GreaterThanOrEqualToThreshold",
            "AlarmActions": [
                "arn:aws:sns:us-east-1:111122223333:NotifyMe"
            ],
            "Namespace": "AWS/EC2",
            "AlarmDescription": "CPU usage exceeds 70 percent",
            "StateReasonData": "{\"version\":\"1.0\",\"queryDate\":
\"2013-10-30T03:03:51.479+0000\",\"startDate\":\"2013-10-30T02:08:00.000+0000\",
\"statistic\":\"Average\",\"period\":300,\"recentDatapoints\":
[40.698,39.612,42.432,39.796,38.816,42.28,42.854,40.088,40.760000000000005,41.316],
\"threshold\":70.0}",
            "Period": 300,
            "StateValue": "OK",
            "Threshold": 70.0,
            "AlarmName": "myHighCpuAlarm2",
            "Dimensions": [
                {
                    "Name": "InstanceId",
                    "Value": "i-0c986c72"
                }
            ],
            "Statistic": "Average",
            "StateReason": "Threshold Crossed: 10 datapoints were not
 greater than or equal to the threshold (70.0). The most recent datapoints:
 [40.760000000000005, 41.316].",
            "InsufficientDataActions": [],
            "OKActions": [],
            "ActionsEnabled": true,
            "MetricName": "CPUUtilization"
        },
        {
            "EvaluationPeriods": 2,
            "AlarmArn": "arn:aws:cloudwatch:us-
east-1:111122223333:alarm:myHighCpuAlarm",
```

```
            "StateUpdatedTimestamp": "2014-04-09T18:59:06.442Z",
            "AlarmConfigurationUpdatedTimestamp": "2014-04-09T22:26:05.958Z",
            "ComparisonOperator": "GreaterThanThreshold",
            "AlarmActions": [
                "arn:aws:sns:us-east-1:111122223333:HighCPUAlarm"
            ],
            "Namespace": "AWS/EC2",
            "AlarmDescription": "CPU usage exceeds 70 percent",
            "StateReasonData": "{\"version\":\"1.0\",\"queryDate\":
\"2014-04-09T18:59:06.419+0000\",\"startDate\":\"2014-04-09T18:44:00.000+0000\",
\"statistic\":\"Average\",\"period\":300,\"recentDatapoints\":[38.958,40.292],
\"threshold\":70.0}",
            "Period": 300,
            "StateValue": "OK",
            "Threshold": 70.0,
            "AlarmName": "myHighCpuAlarm",
            "Dimensions": [
                {
                    "Name": "InstanceId",
                    "Value": "i-0c986c72"
                }
            ],
            "Statistic": "Average",
            "StateReason": "Threshold Crossed: 2 datapoints were not greater than
 the threshold (70.0). The most recent datapoints: [38.958, 40.292].",
            "InsufficientDataActions": [],
            "OKActions": [],
            "ActionsEnabled": false,
            "MetricName": "CPUUtilization"
        }
    ]
}
```

- API の詳細については、AWS CLI コマンドリファレンスの「DescribeAlarmsForMetric」を参照してください。

Java

SDK for Java 2.x

> **Note**
>
> GitHub には、その他のリソースもあります。完全な例を見つけて、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
   public static void checkForMetricAlarm(CloudWatchClient cw, String fileName)
{
       try {
           // Read values from the JSON file.
           JsonParser parser = new JsonFactory().createParser(new
File(fileName));
           com.fasterxml.jackson.databind.JsonNode rootNode = new
ObjectMapper().readTree(parser);
           String customMetricNamespace =
rootNode.findValue("customMetricNamespace").asText();
           String customMetricName =
rootNode.findValue("customMetricName").asText();
           boolean hasAlarm = false;
           int retries = 10;

           DescribeAlarmsForMetricRequest metricRequest =
DescribeAlarmsForMetricRequest.builder()
                   .metricName(customMetricName)
                   .namespace(customMetricNamespace)
                   .build();

           while (!hasAlarm && retries > 0) {
               DescribeAlarmsForMetricResponse response =
cw.describeAlarmsForMetric(metricRequest);
               hasAlarm = response.hasMetricAlarms();
               retries--;
               Thread.sleep(20000);
               System.out.println(".");
           }
           if (!hasAlarm)
               System.out.println("No Alarm state found for " + customMetricName
+ " after 10 retries.");
```

```
            else
                System.out.println("Alarm state found for " + customMetricName +
 ".");

        } catch (CloudWatchException | IOException | InterruptedException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }
```

- API の詳細については、「AWS SDK for Java 2.x API リファレンス」の「DescribeAlarmsForMetric」を参照してください。

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

SDK モジュールとクライアントモジュールをインポートし、API を呼び出します。

```
import { DescribeAlarmsCommand } from "@aws-sdk/client-cloudwatch";
import { client } from "../libs/client.js";

const run = async () => {
  const command = new DescribeAlarmsCommand({
    AlarmNames: [process.env.CLOUDWATCH_ALARM_NAME], // Set the value of
 CLOUDWATCH_ALARM_NAME to the name of an existing alarm.
  });

  try {
    return await client.send(command);
  } catch (err) {
    console.error(err);
  }
};
```

```
export default run();
```

別のモジュールでクライアントを作成し、エクスポートします。

```
import { CloudWatchClient } from "@aws-sdk/client-cloudwatch";

export const client = new CloudWatchClient({});
```

- 詳細については、「AWS SDK for JavaScript デベロッパーガイド」を参照してください。
- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「DescribeAlarmsForMetric」を参照してください。

SDK for JavaScript (v2)

> **Note**
>
> GitHub には、その他のリソースもあります。完全な例を見つけて、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create CloudWatch service object
var cw = new AWS.CloudWatch({ apiVersion: "2010-08-01" });

cw.describeAlarms({ StateValue: "INSUFFICIENT_DATA" }, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    // List the names of all current alarms in the console
    data.MetricAlarms.forEach(function (item, index, array) {
      console.log(item.AlarmName);
    });
  }
});
```

- 詳細については、「AWS SDK for JavaScript デベロッパーガイド」を参照してください。
- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「DescribeAlarmsForMetric」を参照してください。

Kotlin

SDK for Kotlin

> **Note**
>
> GitHub には、その他のリソースもあります。完全な例を見つけて、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
suspend fun checkForMetricAlarm(fileName: String?) {
    // Read values from the JSON file.
    val parser = JsonFactory().createParser(File(fileName))
    val rootNode = ObjectMapper().readTree<JsonNode>(parser)
    val customMetricNamespace =
 rootNode.findValue("customMetricNamespace").asText()
    val customMetricName = rootNode.findValue("customMetricName").asText()
    var hasAlarm = false
    var retries = 10

    val metricRequest = DescribeAlarmsForMetricRequest {
        metricName = customMetricName
        namespace = customMetricNamespace
    }
    CloudWatchClient { region = "us-east-1" }.use { cwClient ->
        while (!hasAlarm && retries > 0) {
            val response = cwClient.describeAlarmsForMetric(metricRequest)
            if (response.metricAlarms?.count()!! > 0) {
                hasAlarm = true
            }
            retries--
            delay(20000)
            println(".")
        }
        if (!hasAlarm) println("No Alarm state found for $customMetricName after
 10 retries.") else println("Alarm state found for $customMetricName.")
    }
```

```
}
```

- API の詳細については、AWS SDK for Kotlin API リファレンスの「DescribeAlarmsForMetric」を参照してください。

Python

SDK for Python (Boto3)

> **Note**
>
> GitHub には、その他のリソースもあります。完全な例を見つけて、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
class CloudWatchWrapper:
    """Encapsulates Amazon CloudWatch functions."""

    def __init__(self, cloudwatch_resource):
        """
        :param cloudwatch_resource: A Boto3 CloudWatch resource.
        """
        self.cloudwatch_resource = cloudwatch_resource


    def get_metric_alarms(self, metric_namespace, metric_name):
        """
        Gets the alarms that are currently watching the specified metric.

        :param metric_namespace: The namespace of the metric.
        :param metric_name: The name of the metric.
        :returns: An iterator that yields the alarms.
        """
        metric = self.cloudwatch_resource.Metric(metric_namespace, metric_name)
        alarm_iter = metric.alarms.all()
        logger.info("Got alarms for metric %s.%s.", metric_namespace,
 metric_name)
        return alarm_iter
```

- API の詳細については、AWS SDK for Python (Boto3) API リファレンスの「DescribeAlarmsForMetric」を参照してください。

Ruby

SDK for Ruby

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
#
# @param cloudwatch_client [Aws::CloudWatch::Client]
#   An initialized CloudWatch client.
# @example
#   describe_metric_alarms(Aws::CloudWatch::Client.new(region: 'us-east-1'))
def describe_metric_alarms(cloudwatch_client)
  response = cloudwatch_client.describe_alarms

  if response.metric_alarms.count.positive?
    response.metric_alarms.each do |alarm|
      puts "-" * 16
      puts "Name:           " + alarm.alarm_name
      puts "State value:    " + alarm.state_value
      puts "State reason:   " + alarm.state_reason
      puts "Metric:         " + alarm.metric_name
      puts "Namespace:      " + alarm.namespace
      puts "Statistic:      " + alarm.statistic
      puts "Period:         " + alarm.period.to_s
      puts "Unit:           " + alarm.unit.to_s
      puts "Eval. periods:  " + alarm.evaluation_periods.to_s
      puts "Threshold:      " + alarm.threshold.to_s
      puts "Comp. operator: " + alarm.comparison_operator

      if alarm.key?(:ok_actions) && alarm.ok_actions.count.positive?
        puts "OK actions:"
        alarm.ok_actions.each do |a|
          puts "  " + a
```

```
        end
      end

      if alarm.key?(:alarm_actions) && alarm.alarm_actions.count.positive?
        puts "Alarm actions:"
        alarm.alarm_actions.each do |a|
          puts "  " + a
        end
      end

      if alarm.key?(:insufficient_data_actions) &&
          alarm.insufficient_data_actions.count.positive?
        puts "Insufficient data actions:"
        alarm.insufficient_data_actions.each do |a|
          puts "  " + a
        end
      end

      puts "Dimensions:"
      if alarm.key?(:dimensions) && alarm.dimensions.count.positive?
        alarm.dimensions.each do |d|
          puts "  Name: " + d.name + ", Value: " + d.value
        end
      else
        puts "  None for this alarm."
      end
    end
  else
    puts "No alarms found."
  end
rescue StandardError => e
  puts "Error getting information about alarms: #{e.message}"
end

# Example usage:
def run_me
  region = ""

  # Print usage information and then stop.
  if ARGV[0] == "--help" || ARGV[0] == "-h"
    puts "Usage:   ruby cw-ruby-example-show-alarms.rb REGION"
    puts "Example: ruby cw-ruby-example-show-alarms.rb us-east-1"
    exit 1
  # If no values are specified at the command prompt, use these default values.
```

```
  elsif ARGV.count.zero?
    region = "us-east-1"
  # Otherwise, use the values as specified at the command prompt.
  else
    region = ARGV[0]
  end

  cloudwatch_client = Aws::CloudWatch::Client.new(region: region)
  puts "Available alarms:"
  describe_metric_alarms(cloudwatch_client)
end

run_me if $PROGRAM_NAME == __FILE__
```

- API の詳細については、「AWS SDK for Ruby API リファレンス」の「DescribeAlarmsForMetric」を参照してください。

AWS SDK デベロッパーガイドとコード例の完全なリストについては、「AWS SDK で CloudWatch を使用する」を参照してください。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK または CLI で **DescribeAnomalyDetectors** を使用する

以下のコード例は、DescribeAnomalyDetectors の使用方法を示しています。

アクション例は、より大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。次のコード例で、このアクションのコンテキストを確認できます。

- メトリクス、ダッシュボード、およびアラームの使用を開始する

.NET

AWS SDK for .NET

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Describe anomaly detectors for a metric and namespace.
    /// </summary>
    /// <param name="metricNamespace">The namespace of the metric.</param>
    /// <param name="metricName">The metric of the anomaly detectors.</param>
    /// <returns>The list of detectors.</returns>
    public async Task<List<AnomalyDetector>> DescribeAnomalyDetectors(string
metricNamespace, string metricName)
    {
        List<AnomalyDetector> detectors = new List<AnomalyDetector>();
        var paginatedDescribeAnomalyDetectors =
_amazonCloudWatch.Paginators.DescribeAnomalyDetectors(
            new DescribeAnomalyDetectorsRequest()
            {
                MetricName = metricName,
                Namespace = metricNamespace
            });

        await foreach (var data in
paginatedDescribeAnomalyDetectors.AnomalyDetectors)
        {
            detectors.Add(data);
        }

        return detectors;
    }
```

- API の詳細については、「AWS SDK for .NET API リファレンス」の「DescribeAnomalyDetectors」を参照してください。

Java

SDK for Java 2.x

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
   public static void describeAnomalyDetectors(CloudWatchClient cw, String
fileName) {
       try {
           // Read values from the JSON file.
           JsonParser parser = new JsonFactory().createParser(new
File(fileName));
           com.fasterxml.jackson.databind.JsonNode rootNode = new
ObjectMapper().readTree(parser);
           String customMetricNamespace =
rootNode.findValue("customMetricNamespace").asText();
           String customMetricName =
rootNode.findValue("customMetricName").asText();
           DescribeAnomalyDetectorsRequest detectorsRequest =
DescribeAnomalyDetectorsRequest.builder()
                   .maxResults(10)
                   .metricName(customMetricName)
                   .namespace(customMetricNamespace)
                   .build();

           DescribeAnomalyDetectorsResponse response =
cw.describeAnomalyDetectors(detectorsRequest);
           List<AnomalyDetector> anomalyDetectorList =
response.anomalyDetectors();
           for (AnomalyDetector detector : anomalyDetectorList) {
               System.out.println("Metric name: " +
detector.singleMetricAnomalyDetector().metricName());
               System.out.println("State: " + detector.stateValue());
           }

       } catch (CloudWatchException | IOException e) {
           System.err.println(e.getMessage());
           System.exit(1);
       }
   }
```

- API の詳細については、「AWS SDK for Java 2.x API リファレンス」の「DescribeAnomalyDetectors」を参照してください。

Kotlin

SDK for Kotlin

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
suspend fun describeAnomalyDetectors(fileName: String) {
    // Read values from the JSON file.
    val parser = JsonFactory().createParser(File(fileName))
    val rootNode = ObjectMapper().readTree<JsonNode>(parser)
    val customMetricNamespace =
 rootNode.findValue("customMetricNamespace").asText()
    val customMetricName = rootNode.findValue("customMetricName").asText()

    val detectorsRequest = DescribeAnomalyDetectorsRequest {
        maxResults = 10
        metricName = customMetricName
        namespace = customMetricNamespace
    }
    CloudWatchClient { region = "us-east-1" }.use { cwClient ->
        val response = cwClient.describeAnomalyDetectors(detectorsRequest)
        response.anomalyDetectors?.forEach { detector ->
            println("Metric name:
 ${detector.singleMetricAnomalyDetector?.metricName}")
            println("State: ${detector.stateValue}")
        }
    }
}
```

- API の詳細については、「AWS SDK for Kotlin API リファレンス」の「DescribeAnomalyDetectors」を参照してください。

AWS SDK デベロッパーガイドとコード例の完全なリストについては、「AWS SDK で CloudWatch を使用する」を参照してください。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK または CLI で **DisableAlarmActions** を使用する

以下のコード例は、DisableAlarmActions の使用方法を示しています。

アクション例は、より大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。次のコード例で、このアクションのコンテキストを確認できます。

- アラームの使用を開始
- メトリクスとアラームを管理する

.NET

AWS SDK for .NET

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Disable the actions for a list of alarms from CloudWatch.
    /// </summary>
    /// <param name="alarmNames">A list of names of alarms.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DisableAlarmActions(List<string> alarmNames)
    {
        var disableAlarmActionsResult = await
_amazonCloudWatch.DisableAlarmActionsAsync(
            new DisableAlarmActionsRequest()
            {
                AlarmNames = alarmNames
            });

        return disableAlarmActionsResult.HttpStatusCode == HttpStatusCode.OK;
    }
```

- API 詳細については、「AWS SDK for .NET API リファレンス」の「DisableAlarmActions」を参照してください。

C++

SDK for C++

> **Note**
>
> GitHub には、その他のリソースもあります。完全な例を見つけて、AWS コード例リポジトリでの設定と実行の方法を確認してください。

必要なファイルを含めます。

```
#include <aws/core/Aws.h>
#include <aws/monitoring/CloudWatchClient.h>
#include <aws/monitoring/model/DisableAlarmActionsRequest.h>
#include <iostream>
```

アラームアクションを無効化します。

```
        Aws::CloudWatch::CloudWatchClient cw;

        Aws::CloudWatch::Model::DisableAlarmActionsRequest
 disableAlarmActionsRequest;
        disableAlarmActionsRequest.AddAlarmNames(alarm_name);

        auto disableAlarmActionsOutcome =
 cw.DisableAlarmActions(disableAlarmActionsRequest);
        if (!disableAlarmActionsOutcome.IsSuccess())
        {
            std::cout << "Failed to disable actions for alarm " << alarm_name <<
                ": " << disableAlarmActionsOutcome.GetError().GetMessage() <<
                std::endl;
        }
        else
        {
            std::cout << "Successfully disabled actions for alarm " <<
                alarm_name << std::endl;
        }
```

- API 詳細については、「AWS SDK for C++ API リファレンス」の「DisableAlarmActions」を参照してください。

CLI

AWS CLI

アラームのアクションを無効化するには

次の例は、disable-alarm-actions コマンドを使用して、「myalarm」という名前のアラームのアクションをすべて無効化します。

```
aws cloudwatch disable-alarm-actions --alarm-names myalarm
```

正常に完了すると、このコマンドはプロンプトに戻ります。

- API の詳細については、AWS CLI コマンドリファレンスの「DisableAlarmActions」を参照してください。

Java

SDK for Java 2.x

> **Note**
>
> GitHub には、その他のリソースもあります。完全な例を見つけて、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.cloudwatch.CloudWatchClient;
import software.amazon.awssdk.services.cloudwatch.model.CloudWatchException;
import
 software.amazon.awssdk.services.cloudwatch.model.DisableAlarmActionsRequest;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
```

```
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class DisableAlarmActions {
    public static void main(String[] args) {
        final String usage = """

                Usage:
                  <alarmName>

                Where:
                  alarmName - An alarm name to disable (for example, MyAlarm).
                """;

        if (args.length != 1) {
            System.out.println(usage);
            System.exit(1);
        }

        String alarmName = args[0];
        Region region = Region.US_EAST_1;
        CloudWatchClient cw = CloudWatchClient.builder()
                .region(region)
                .build();

        disableActions(cw, alarmName);
        cw.close();
    }

    public static void disableActions(CloudWatchClient cw, String alarmName) {
        try {
            DisableAlarmActionsRequest request =
 DisableAlarmActionsRequest.builder()
                    .alarmNames(alarmName)
                    .build();

            cw.disableAlarmActions(request);
            System.out.printf("Successfully disabled actions on alarm %s",
 alarmName);

        } catch (CloudWatchException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
```

```
    }
}
```

- API 詳細については、「AWS SDK for Java 2.x API リファレンス」の「DisableAlarmActions」を参照してください。

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

SDK モジュールとクライアントモジュールをインポートし、API を呼び出します。

```
import { DisableAlarmActionsCommand } from "@aws-sdk/client-cloudwatch";
import { client } from "../libs/client.js";

const run = async () => {
  const command = new DisableAlarmActionsCommand({
    AlarmNames: process.env.CLOUDWATCH_ALARM_NAME, // Set the value of
 CLOUDWATCH_ALARM_NAME to the name of an existing alarm.
  });

  try {
    return await client.send(command);
  } catch (err) {
    console.error(err);
  }
};

export default run();
```

別のモジュールでクライアントを作成し、エクスポートします。

```
import { CloudWatchClient } from "@aws-sdk/client-cloudwatch";
```

```
export const client = new CloudWatchClient({});
```

- 詳細については、「AWS SDK for JavaScript デベロッパーガイド」を参照してください。
- API 詳細については、「AWS SDK for JavaScript API リファレンス」の「DisableAlarmActions」を参照してください。

SDK for JavaScript (v2)

> **Note**
>
> GitHub には、その他のリソースもあります。完全な例を見つけて、AWS コード例リポジトリでの設定と実行の方法を確認してください。

SDK モジュールとクライアントモジュールをインポートし、API を呼び出します。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create CloudWatch service object
var cw = new AWS.CloudWatch({ apiVersion: "2010-08-01" });

cw.disableAlarmActions(
  { AlarmNames: ["Web_Server_CPU_Utilization"] },
  function (err, data) {
    if (err) {
      console.log("Error", err);
    } else {
      console.log("Success", data);
    }
  }
);
```

- 詳細については、AWS SDK for JavaScript デベロッパーガイドを参照してください。
- API 詳細については、「AWS SDK for JavaScript API リファレンス」の「DisableAlarmActions」を参照してください。

Kotlin

SDK for Kotlin

> **Note**
>
> GitHub には、その他のリソースもあります。完全な例を見つけて、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
suspend fun disableActions(alarmName: String) {

    val request = DisableAlarmActionsRequest {
        alarmNames = listOf(alarmName)
    }
    CloudWatchClient { region = "us-east-1" }.use { cwClient ->
        cwClient.disableAlarmActions(request)
        println("Successfully disabled actions on alarm $alarmName")
    }
}
```

- API の詳細については、AWS SDK for Kotlin API リファレンスの「DisableAlarmActions」を参照してください。

Python

SDK for Python (Boto3)

> **Note**
>
> GitHub には、その他のリソースもあります。完全な例を見つけて、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
class CloudWatchWrapper:
    """Encapsulates Amazon CloudWatch functions."""

    def __init__(self, cloudwatch_resource):
```

```
        """
        :param cloudwatch_resource: A Boto3 CloudWatch resource.
        """
        self.cloudwatch_resource = cloudwatch_resource


    def enable_alarm_actions(self, alarm_name, enable):
        """
        Enables or disables actions on the specified alarm. Alarm actions can be
        used to send notifications or automate responses when an alarm enters a
        particular state.

        :param alarm_name: The name of the alarm.
        :param enable: When True, actions are enabled for the alarm. Otherwise,
they
                       disabled.
        """
        try:
            alarm = self.cloudwatch_resource.Alarm(alarm_name)
            if enable:
                alarm.enable_actions()
            else:
                alarm.disable_actions()
            logger.info(
                "%s actions for alarm %s.",
                "Enabled" if enable else "Disabled",
                alarm_name,
            )
        except ClientError:
            logger.exception(
                "Couldn't %s actions alarm %s.",
                "enable" if enable else "disable",
                alarm_name,
            )
            raise
```

- API の詳細については、AWS SDK for Python (Boto3) API リファレンスの「DisableAlarmActions」を参照してください。

Ruby

SDK for Ruby

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
# Disables an alarm in Amazon CloudWatch.
#
# Prerequisites.
#
# - The alarm to disable.
#
# @param cloudwatch_client [Aws::CloudWatch::Client]
#   An initialized CloudWatch client.
# @param alarm_name [String] The name of the alarm to disable.
# @return [Boolean] true if the alarm was disabled; otherwise, false.
# @example
#   exit 1 unless alarm_actions_disabled?(
#     Aws::CloudWatch::Client.new(region: 'us-east-1'),
#     'ObjectsInBucket'
#   )
def alarm_actions_disabled?(cloudwatch_client, alarm_name)
  cloudwatch_client.disable_alarm_actions(alarm_names: [alarm_name])
  return true
rescue StandardError => e
  puts "Error disabling alarm actions: #{e.message}"
  return false
end

# Example usage:
def run_me
  alarm_name = "ObjectsInBucket"
  alarm_description = "Objects exist in this bucket for more than 1 day."
  metric_name = "NumberOfObjects"
  # Notify this Amazon Simple Notification Service (Amazon SNS) topic when
  # the alarm transitions to the ALARM state.
  alarm_actions = ["arn:aws:sns:us-
east-1:111111111111:Default_CloudWatch_Alarms_Topic"]
```

```
  namespace = "AWS/S3"
  statistic = "Average"
  dimensions = [
    {
      name: "BucketName",
      value: "doc-example-bucket"
    },
    {
      name: "StorageType",
      value: "AllStorageTypes"
    }
  ]
  period = 86_400 # Daily (24 hours * 60 minutes * 60 seconds = 86400 seconds).
  unit = "Count"
  evaluation_periods = 1 # More than one day.
  threshold = 1 # One object.
  comparison_operator = "GreaterThanThreshold" # More than one object.
  # Replace us-west-2 with the AWS Region you're using for Amazon CloudWatch.
  region = "us-east-1"

  cloudwatch_client = Aws::CloudWatch::Client.new(region: region)

  if alarm_created_or_updated?(
    cloudwatch_client,
    alarm_name,
    alarm_description,
    metric_name,
    alarm_actions,
    namespace,
    statistic,
    dimensions,
    period,
    unit,
    evaluation_periods,
    threshold,
    comparison_operator
  )
    puts "Alarm '#{alarm_name}' created or updated."
  else
    puts "Could not create or update alarm '#{alarm_name}'."
  end

  if alarm_actions_disabled?(cloudwatch_client, alarm_name)
    puts "Alarm '#{alarm_name}' disabled."
```

```
  else
    puts "Could not disable alarm '#{alarm_name}'."
  end
end

run_me if $PROGRAM_NAME == __FILE__
```

- API 詳細については、「AWS SDK for Ruby API リファレンス」の「DisableAlarmActions」を参照してください。

SAP ABAP

SDK for SAP ABAP

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
    "Disables actions on the specified alarm. "
    TRY.
        lo_cwt->disablealarmactions(
          it_alarmnames = it_alarm_names
        ).
        MESSAGE 'Alarm actions disabled.' TYPE 'I'.
      CATCH /aws1/cx_rt_service_generic INTO DATA(lo_exception).
        DATA(lv_error) = |"{ lo_exception->av_err_code }" - { lo_exception-
>av_err_msg }|.
        MESSAGE lv_error TYPE 'E'.
    ENDTRY.
```

- API の詳細については、「AWS SDK for SAP ABAP API リファレンス」の「DisableAlarmActions」を参照してください。

AWS SDK デベロッパーガイドとコード例の完全なリストについては、「AWS SDK で CloudWatch を使用する」を参照してください。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK または CLI で **EnableAlarmActions** を使用する

以下のコード例は、EnableAlarmActions の使用方法を示しています。

アクション例は、より大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。次のコード例で、このアクションのコンテキストを確認できます。

- メトリクスとアラームを管理する

.NET

AWS SDK for .NET

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Enable the actions for a list of alarms from CloudWatch.
    /// </summary>
    /// <param name="alarmNames">A list of names of alarms.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> EnableAlarmActions(List<string> alarmNames)
    {
        var enableAlarmActionsResult = await
_amazonCloudWatch.EnableAlarmActionsAsync(
            new EnableAlarmActionsRequest()
            {
                AlarmNames = alarmNames
            });

        return enableAlarmActionsResult.HttpStatusCode == HttpStatusCode.OK;
    }
```

- API の詳細については、「AWS SDK for .NET API リファレンス」の「EnableAlarmActions」を参照してください。

C++

SDK for C++

> **Note**
>
> GitHub には、その他のリソースもあります。完全な例を見つけて、AWS コード例リポジトリでの設定と実行の方法を確認してください。

必要なファイルを含めます。

```
#include <aws/core/Aws.h>
#include <aws/monitoring/CloudWatchClient.h>
#include <aws/monitoring/model/EnableAlarmActionsRequest.h>
#include <aws/monitoring/model/PutMetricAlarmRequest.h>
#include <iostream>
```

アラームアクションを有効化します。

```
    Aws::CloudWatch::CloudWatchClient cw;
    Aws::CloudWatch::Model::PutMetricAlarmRequest request;
    request.SetAlarmName(alarm_name);
    request.SetComparisonOperator(
        Aws::CloudWatch::Model::ComparisonOperator::GreaterThanThreshold);
    request.SetEvaluationPeriods(1);
    request.SetMetricName("CPUUtilization");
    request.SetNamespace("AWS/EC2");
    request.SetPeriod(60);
    request.SetStatistic(Aws::CloudWatch::Model::Statistic::Average);
    request.SetThreshold(70.0);
    request.SetActionsEnabled(false);
    request.SetAlarmDescription("Alarm when server CPU exceeds 70%");
    request.SetUnit(Aws::CloudWatch::Model::StandardUnit::Seconds);
    request.AddAlarmActions(actionArn);

    Aws::CloudWatch::Model::Dimension dimension;
```

```
    dimension.SetName("InstanceId");
    dimension.SetValue(instanceId);
    request.AddDimensions(dimension);

    auto outcome = cw.PutMetricAlarm(request);
    if (!outcome.IsSuccess())
    {
        std::cout << "Failed to create CloudWatch alarm:" <<
            outcome.GetError().GetMessage() << std::endl;
        return;
    }

    Aws::CloudWatch::Model::EnableAlarmActionsRequest enable_request;
    enable_request.AddAlarmNames(alarm_name);

    auto enable_outcome = cw.EnableAlarmActions(enable_request);
    if (!enable_outcome.IsSuccess())
    {
        std::cout << "Failed to enable alarm actions:" <<
            enable_outcome.GetError().GetMessage() << std::endl;
        return;
    }

    std::cout << "Successfully created alarm " << alarm_name <<
        " and enabled actions on it." << std::endl;
```

- API の詳細については、「AWS SDK for C++ API リファレンス」の「EnableAlarmActions」を参照してください。

CLI

AWS CLI

アラームのすべてのアクションを有効化するには

次の例は、enable-alarm-actions コマンドを使用して、「myalarm」という名前のアラームのアクションをすべて有効化します。

```
aws cloudwatch enable-alarm-actions --alarm-names myalarm
```

正常に完了すると、このコマンドはプロンプトに戻ります。

- API の詳細については、AWS CLI コマンドリファレンスの「EnableAlarmActions」を参照してください。

Java

SDK for Java 2.x

> **Note**
>
> GitHub には、その他のリソースもあります。完全な例を見つけて、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.cloudwatch.CloudWatchClient;
import software.amazon.awssdk.services.cloudwatch.model.CloudWatchException;
import
 software.amazon.awssdk.services.cloudwatch.model.EnableAlarmActionsRequest;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class EnableAlarmActions {
    public static void main(String[] args) {
        final String usage = """

                Usage:
                  <alarmName>

                Where:
                  alarmName - An alarm name to enable (for example, MyAlarm).
                """;

        if (args.length != 1) {
            System.out.println(usage);
```

```
            System.exit(1);
        }

        String alarm = args[0];
        Region region = Region.US_EAST_1;
        CloudWatchClient cw = CloudWatchClient.builder()
                .region(region)
                .build();

        enableActions(cw, alarm);
        cw.close();
    }

    public static void enableActions(CloudWatchClient cw, String alarm) {
        try {
            EnableAlarmActionsRequest request =
 EnableAlarmActionsRequest.builder()
                    .alarmNames(alarm)
                    .build();

            cw.enableAlarmActions(request);
            System.out.printf("Successfully enabled actions on alarm %s", alarm);

        } catch (CloudWatchException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }
}
```

- API の詳細については、「AWS SDK for Java 2.x API リファレンス」の「EnableAlarmActions」を参照してください。

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

SDK モジュールとクライアントモジュールをインポートし、API を呼び出します。

```
import { EnableAlarmActionsCommand } from "@aws-sdk/client-cloudwatch";
import { client } from "../libs/client.js";

const run = async () => {
  const command = new EnableAlarmActionsCommand({
    AlarmNames: [process.env.CLOUDWATCH_ALARM_NAME], // Set the value of
 CLOUDWATCH_ALARM_NAME to the name of an existing alarm.
  });

  try {
    return await client.send(command);
  } catch (err) {
    console.error(err);
  }
};

export default run();
```

別のモジュールでクライアントを作成し、エクスポートします。

```
import { CloudWatchClient } from "@aws-sdk/client-cloudwatch";

export const client = new CloudWatchClient({});
```

- 詳細については、「AWS SDK for JavaScript デベロッパーガイド」を参照してください。
- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「EnableAlarmActions」を参照してください。

SDK for JavaScript (v2)

> **Note**
>
> GitHub には、その他のリソースもあります。完全な例を見つけて、AWS コード例リポジトリでの設定と実行の方法を確認してください。

SDK モジュールとクライアントモジュールをインポートし、API を呼び出します。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create CloudWatch service object
var cw = new AWS.CloudWatch({ apiVersion: "2010-08-01" });

var params = {
  AlarmName: "Web_Server_CPU_Utilization",
  ComparisonOperator: "GreaterThanThreshold",
  EvaluationPeriods: 1,
  MetricName: "CPUUtilization",
  Namespace: "AWS/EC2",
  Period: 60,
  Statistic: "Average",
  Threshold: 70.0,
  ActionsEnabled: true,
  AlarmActions: ["ACTION_ARN"],
  AlarmDescription: "Alarm when server CPU exceeds 70%",
  Dimensions: [
    {
      Name: "InstanceId",
      Value: "INSTANCE_ID",
    },
  ],
  Unit: "Percent",
};

cw.putMetricAlarm(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
```

```
    console.log("Alarm action added", data);
    var paramsEnableAlarmAction = {
      AlarmNames: [params.AlarmName],
    };
    cw.enableAlarmActions(paramsEnableAlarmAction, function (err, data) {
      if (err) {
        console.log("Error", err);
      } else {
        console.log("Alarm action enabled", data);
      }
    });
  }
});
```

- 詳細については、AWS SDK for JavaScript デベロッパーガイドを参照してください。
- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「EnableAlarmActions」を参照してください。

Kotlin

SDK for Kotlin

> **Note**
>
> GitHub には、その他のリソースもあります。完全な例を見つけて、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
suspend fun enableActions(alarm: String) {

    val request = EnableAlarmActionsRequest {
        alarmNames = listOf(alarm)
    }

    CloudWatchClient { region = "us-east-1" }.use { cwClient ->
        cwClient.enableAlarmActions(request)
        println("Successfully enabled actions on alarm $alarm")
    }
}
```

- API の詳細については、AWS SDK for Kotlin API リファレンスの「EnableAlarmActions」を参照してください。

Python

SDK for Python (Boto3)

> **Note**
>
> GitHub には、その他のリソースもあります。完全な例を見つけて、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
class CloudWatchWrapper:
    """Encapsulates Amazon CloudWatch functions."""

    def __init__(self, cloudwatch_resource):
        """
        :param cloudwatch_resource: A Boto3 CloudWatch resource.
        """
        self.cloudwatch_resource = cloudwatch_resource


    def enable_alarm_actions(self, alarm_name, enable):
        """
        Enables or disables actions on the specified alarm. Alarm actions can be
        used to send notifications or automate responses when an alarm enters a
        particular state.

        :param alarm_name: The name of the alarm.
        :param enable: When True, actions are enabled for the alarm. Otherwise,
 they
                       disabled.
        """
        try:
            alarm = self.cloudwatch_resource.Alarm(alarm_name)
            if enable:
                alarm.enable_actions()
            else:
```

```
                alarm.disable_actions()
            logger.info(
                "%s actions for alarm %s.",
                "Enabled" if enable else "Disabled",
                alarm_name,
            )
        except ClientError:
            logger.exception(
                "Couldn't %s actions alarm %s.",
                "enable" if enable else "disable",
                alarm_name,
            )
            raise
```

- API の詳細については、AWS SDK for Python (Boto3) API リファレンスの「EnableAlarmActions」を参照してください。

SAP ABAP

SDK for SAP ABAP

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
    "Enable actions on the specified alarm."
    TRY.
        lo_cwt->enablealarmactions(
          it_alarmnames = it_alarm_names
        ).
        MESSAGE 'Alarm actions enabled.' TYPE 'I'.
      CATCH /aws1/cx_rt_service_generic INTO DATA(lo_exception).
        DATA(lv_error) = |"{ lo_exception->av_err_code }" - { lo_exception-
>av_err_msg }|.
        MESSAGE lv_error TYPE 'E'.
    ENDTRY.
```

- API の詳細については、「AWS SDK for SAP ABAP API リファレンス」の「EnableAlarmActions」を参照してください。

AWS SDK デベロッパーガイドとコード例の完全なリストについては、「AWS SDK で CloudWatch を使用する」を参照してください。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK または CLI で **GetDashboard** を使用する

以下のコード例は、GetDashboard の使用方法を示しています。

.NET

AWS SDK for .NET

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
/// <summary>
/// Get information on a dashboard.
/// </summary>
/// <param name="dashboardName">The name of the dashboard.</param>
/// <returns>A JSON object with dashboard information.</returns>
public async Task<string> GetDashboard(string dashboardName)
{
    var dashboardResponse = await _amazonCloudWatch.GetDashboardAsync(
        new GetDashboardRequest()
        {
            DashboardName = dashboardName
        });

    return dashboardResponse.DashboardBody;
}
```

- API の詳細については、AWS SDK for .NET API リファレンスの「GetDashboard」を参照してください。

PowerShell

Tools for PowerShell

例 1: 指定されたダッシュボードの本体である arn を返します。

```
Get-CWDashboard -DashboardName Dashboard1
```

出力:

```
DashboardArn                                                DashboardBody
------------                                                -------------
arn:aws:cloudwatch::123456789012:dashboard/Dashboard1 {...
```

- API の詳細については、「AWS Tools for PowerShell コマンドレットリファレンス」の「GetDashboard」を参照してください。

AWS SDK デベロッパーガイドとコード例の完全なリストについては、「AWS SDK で CloudWatch を使用する」を参照してください。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK または CLI で **GetMetricData** を使用する

以下のコード例は、GetMetricData の使用方法を示しています。

アクション例は、より大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。次のコード例で、このアクションのコンテキストを確認できます。

- メトリクス、ダッシュボード、およびアラームの使用を開始する

.NET

AWS SDK for .NET

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Get data for CloudWatch metrics.
    /// </summary>
    /// <param name="minutesOfData">The number of minutes of data to include.</
param>
    /// <param name="useDescendingTime">True to return the data descending by
 time.</param>
    /// <param name="endDateUtc">The end date for the data, in UTC.</param>
    /// <param name="maxDataPoints">The maximum data points to include.</param>
    /// <param name="dataQueries">Optional data queries to include.</param>
    /// <returns>A list of the requested metric data.</returns>
    public async Task<List<MetricDataResult>> GetMetricData(int minutesOfData,
 bool useDescendingTime, DateTime? endDateUtc = null,
        int maxDataPoints = 0, List<MetricDataQuery>? dataQueries = null)
    {
        var metricData = new List<MetricDataResult>();
        // If no end time is provided, use the current time for the end time.
        endDateUtc ??= DateTime.UtcNow;
        var timeZoneOffset =
 TimeZoneInfo.Local.GetUtcOffset(endDateUtc.Value.ToLocalTime());
        var startTimeUtc = endDateUtc.Value.AddMinutes(-minutesOfData);
        // The timezone string should be in the format +0000, so use the timezone
 offset to format it correctly.
        var timeZoneString = $"{timeZoneOffset.Hours:D2}
{timeZoneOffset.Minutes:D2}";
        var paginatedMetricData = _amazonCloudWatch.Paginators.GetMetricData(
            new GetMetricDataRequest()
            {
                StartTimeUtc = startTimeUtc,
                EndTimeUtc = endDateUtc.Value,
                LabelOptions = new LabelOptions { Timezone = timeZoneString },
```

```
                ScanBy = useDescendingTime ? ScanBy.TimestampDescending :
ScanBy.TimestampAscending,
                MaxDatapoints = maxDataPoints,
                MetricDataQueries = dataQueries,
            });

        await foreach (var data in paginatedMetricData.MetricDataResults)
        {
            metricData.Add(data);
        }
        return metricData;
    }
```

- API の詳細については、「AWS SDK for .NET API リファレンス」の「GetMetricData」を参照してください。

Java

SDK for Java 2.x

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
    public static void getCustomMetricData(CloudWatchClient cw, String fileName)
{
        try {
            // Read values from the JSON file.
            JsonParser parser = new JsonFactory().createParser(new
File(fileName));
            com.fasterxml.jackson.databind.JsonNode rootNode = new
ObjectMapper().readTree(parser);
            String customMetricNamespace =
rootNode.findValue("customMetricNamespace").asText();
            String customMetricName =
rootNode.findValue("customMetricName").asText();

            // Set the date.
```

```
            Instant nowDate = Instant.now();

            long hours = 1;
            long minutes = 30;
            Instant date2 = nowDate.plus(hours, ChronoUnit.HOURS).plus(minutes,
                    ChronoUnit.MINUTES);

            Metric met = Metric.builder()
                    .metricName(customMetricName)
                    .namespace(customMetricNamespace)
                    .build();

            MetricStat metStat = MetricStat.builder()
                    .stat("Maximum")
                    .period(1)
                    .metric(met)
                    .build();

            MetricDataQuery dataQUery = MetricDataQuery.builder()
                    .metricStat(metStat)
                    .id("foo2")
                    .returnData(true)
                    .build();

            List<MetricDataQuery> dq = new ArrayList<>();
            dq.add(dataQUery);

            GetMetricDataRequest getMetReq = GetMetricDataRequest.builder()
                    .maxDatapoints(10)
                    .scanBy(ScanBy.TIMESTAMP_DESCENDING)
                    .startTime(nowDate)
                    .endTime(date2)
                    .metricDataQueries(dq)
                    .build();

            GetMetricDataResponse response = cw.getMetricData(getMetReq);
            List<MetricDataResult> data = response.metricDataResults();
            for (MetricDataResult item : data) {
                System.out.println("The label is " + item.label());
                System.out.println("The status code is " +
item.statusCode().toString());
            }

        } catch (CloudWatchException | IOException e) {
```

```
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }
```

- API の詳細については、「AWS SDK for Java 2.x API リファレンス」の「GetMetricData」を参照してください。

Kotlin

SDK for Kotlin

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
suspend fun getCustomMetricData(fileName: String) {
    // Read values from the JSON file.
    val parser = JsonFactory().createParser(File(fileName))
    val rootNode = ObjectMapper().readTree<JsonNode>(parser)
    val customMetricNamespace =
 rootNode.findValue("customMetricNamespace").asText()
    val customMetricName = rootNode.findValue("customMetricName").asText()

    // Set the date.
    val nowDate = Instant.now()
    val hours: Long = 1
    val minutes: Long = 30
    val date2 = nowDate.plus(hours, ChronoUnit.HOURS).plus(
        minutes,
        ChronoUnit.MINUTES
    )

    val met = Metric {
        metricName = customMetricName
        namespace = customMetricNamespace
    }
```

```
    val metStat = MetricStat {
        stat = "Maximum"
        period = 1
        metric = met
    }

    val dataQUery = MetricDataQuery {
        metricStat = metStat
        id = "foo2"
        returnData = true
    }

    val dq = ArrayList<MetricDataQuery>()
    dq.add(dataQUery)
    val getMetReq = GetMetricDataRequest {
        maxDatapoints = 10
        scanBy = ScanBy.TimestampDescending
        startTime = aws.smithy.kotlin.runtime.time.Instant(nowDate)
        endTime = aws.smithy.kotlin.runtime.time.Instant(date2)
        metricDataQueries = dq
    }

    CloudWatchClient { region = "us-east-1" }.use { cwClient ->
        val response = cwClient.getMetricData(getMetReq)
        response.metricDataResults?.forEach { item ->
            println("The label is ${item.label}")
            println("The status code is ${item.statusCode}")
        }
    }
}
```

- API の詳細については、「AWS SDK for Kotlin API リファレンス」の「GetMetricData」を参照してください。

AWS SDK デベロッパーガイドとコード例の完全なリストについては、「AWS SDK で CloudWatch を使用する」を参照してください。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK または CLI で **GetMetricStatistics** を使用する

以下のコード例は、GetMetricStatistics の使用方法を示しています。

アクション例は、より大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。次のコード例で、このアクションのコンテキストを確認できます。

- メトリクス、ダッシュボード、およびアラームの使用を開始する
- メトリクスとアラームを管理する

.NET

AWS SDK for .NET

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Get billing statistics using a call to a wrapper class.
    /// </summary>
    /// <returns>A collection of billing statistics.</returns>
    private static async Task<List<Datapoint>> SetupBillingStatistics()
    {
        // Make a request for EstimatedCharges with a period of one day for the
past seven days.
        var billingStatistics = await _cloudWatchWrapper.GetMetricStatistics(
            "AWS/Billing",
            "EstimatedCharges",
            new List<string>() { "Maximum" },
            new List<Dimension>() { new Dimension { Name = "Currency", Value =
"USD" } },
            7,
            86400);

        billingStatistics = billingStatistics.OrderBy(n => n.Timestamp).ToList();

        return billingStatistics;
    }

    /// <summary>
    /// Wrapper to get statistics for a specific CloudWatch metric.
    /// </summary>
```

```
    /// <param name="metricNamespace">The namespace of the metric.</param>
    /// <param name="metricName">The name of the metric.</param>
    /// <param name="statistics">The list of statistics to include.</param>
    /// <param name="dimensions">The list of dimensions to include.</param>
    /// <param name="days">The number of days in the past to include.</param>
    /// <param name="period">The period for the data.</param>
    /// <returns>A list of DataPoint objects for the statistics.</returns>
    public async Task<List<Datapoint>> GetMetricStatistics(string
metricNamespace,
        string metricName, List<string> statistics, List<Dimension> dimensions,
int days, int period)
    {
        var metricStatistics = await _amazonCloudWatch.GetMetricStatisticsAsync(
            new GetMetricStatisticsRequest()
            {
                Namespace = metricNamespace,
                MetricName = metricName,
                Dimensions = dimensions,
                Statistics = statistics,
                StartTimeUtc = DateTime.UtcNow.AddDays(-days),
                EndTimeUtc = DateTime.UtcNow,
                Period = period
            });

        return metricStatistics.Datapoints;
    }
```

- API の詳細については、「AWS SDK for .NET API リファレンス」の「GetMetricStatistics」を参照してください。

CLI

AWS CLI

EC2 インスタンスあたりの CPU 使用率を取得するには

次の例は、`get-metric-statistics` コマンドを使用して、ID i-abcdef の EC2 インスタンスの CPU 使用率を取得します。

```
aws cloudwatch get-metric-statistics --metric-name CPUUtilization --start-time
 2014-04-08T23:18:00Z --end-time 2014-04-09T23:18:00Z --period 3600 --namespace
 AWS/EC2 --statistics Maximum --dimensions Name=InstanceId,Value=i-abcdef
```

出力:

```
{
    "Datapoints": [
        {
            "Timestamp": "2014-04-09T11:18:00Z",
            "Maximum": 44.79,
            "Unit": "Percent"
        },
        {
            "Timestamp": "2014-04-09T20:18:00Z",
            "Maximum": 47.92,
            "Unit": "Percent"
        },
        {
            "Timestamp": "2014-04-09T19:18:00Z",
            "Maximum": 50.85,
            "Unit": "Percent"
        },
        {
            "Timestamp": "2014-04-09T09:18:00Z",
            "Maximum": 47.92,
            "Unit": "Percent"
        },
        {
            "Timestamp": "2014-04-09T03:18:00Z",
            "Maximum": 76.84,
            "Unit": "Percent"
        },
        {
            "Timestamp": "2014-04-09T21:18:00Z",
            "Maximum": 48.96,
            "Unit": "Percent"
        },
        {
            "Timestamp": "2014-04-09T14:18:00Z",
            "Maximum": 47.92,
            "Unit": "Percent"
        },
```

```
        {
            "Timestamp": "2014-04-09T08:18:00Z",
            "Maximum": 47.92,
            "Unit": "Percent"
        },
        {
            "Timestamp": "2014-04-09T16:18:00Z",
            "Maximum": 45.55,
            "Unit": "Percent"
        },
        {
            "Timestamp": "2014-04-09T06:18:00Z",
            "Maximum": 47.92,
            "Unit": "Percent"
        },
        {
            "Timestamp": "2014-04-09T13:18:00Z",
            "Maximum": 45.08,
            "Unit": "Percent"
        },
        {
            "Timestamp": "2014-04-09T05:18:00Z",
            "Maximum": 47.92,
            "Unit": "Percent"
        },
        {
            "Timestamp": "2014-04-09T18:18:00Z",
            "Maximum": 46.88,
            "Unit": "Percent"
        },
        {
            "Timestamp": "2014-04-09T17:18:00Z",
            "Maximum": 52.08,
            "Unit": "Percent"
        },
        {
            "Timestamp": "2014-04-09T07:18:00Z",
            "Maximum": 47.92,
            "Unit": "Percent"
        },
        {
            "Timestamp": "2014-04-09T02:18:00Z",
            "Maximum": 51.23,
            "Unit": "Percent"
```

```
        },
        {
            "Timestamp": "2014-04-09T12:18:00Z",
            "Maximum": 47.67,
            "Unit": "Percent"
        },
        {
            "Timestamp": "2014-04-08T23:18:00Z",
            "Maximum": 46.88,
            "Unit": "Percent"
        },
        {
            "Timestamp": "2014-04-09T10:18:00Z",
            "Maximum": 51.91,
            "Unit": "Percent"
        },
        {
            "Timestamp": "2014-04-09T04:18:00Z",
            "Maximum": 47.13,
            "Unit": "Percent"
        },
        {
            "Timestamp": "2014-04-09T15:18:00Z",
            "Maximum": 48.96,
            "Unit": "Percent"
        },
        {
            "Timestamp": "2014-04-09T00:18:00Z",
            "Maximum": 48.16,
            "Unit": "Percent"
        },
        {
            "Timestamp": "2014-04-09T01:18:00Z",
            "Maximum": 49.18,
            "Unit": "Percent"
        }
    ],
    "Label": "CPUUtilization"
}
```

複数のディメンションを指定する

次の例は、複数のディメンションを指定する方法を示しています。各ディメンションは名前/値のペアとして指定され、名前と値の間にはカンマが入ります。複数のディメンションはスペースで区切ります。また、1 つのメトリクスに複数のディメンションを含む場合は、定義されているディメンションごとに値を指定する必要があります。

get-metric-statistics コマンドのその他の使用例については、「Amazon CloudWatch ユーザーガイド」の「メトリクスの統計を取得する」を参照してください。

```
aws cloudwatch get-metric-statistics --metric-name Buffers --
namespace MyNameSpace --dimensions Name=InstanceID,Value=i-abcdef
 Name=InstanceType,Value=m1.small --start-time 2016-10-15T04:00:00Z --end-time
 2016-10-19T07:00:00Z --statistics Average --period 60
```

- API の詳細については、AWS CLI コマンドリファレンスの「GetMetricStatistics」を参照してください。

Java

SDK for Java 2.x

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
   public static void getAndDisplayMetricStatistics(CloudWatchClient cw, String
nameSpace, String metVal,
           String metricOption, String date, Dimension myDimension) {
       try {
           Instant start = Instant.parse(date);
           Instant endDate = Instant.now();

           GetMetricStatisticsRequest statisticsRequest =
GetMetricStatisticsRequest.builder()
                   .endTime(endDate)
                   .startTime(start)
                   .dimensions(myDimension)
                   .metricName(metVal)
                   .namespace(nameSpace)
```

```
                    .period(86400)
                    .statistics(Statistic.fromValue(metricOption))
                    .build();

            GetMetricStatisticsResponse response =
 cw.getMetricStatistics(statisticsRequest);
            List<Datapoint> data = response.datapoints();
            if (!data.isEmpty()) {
                for (Datapoint datapoint : data) {
                    System.out
                            .println("Timestamp: " + datapoint.timestamp() + "
 Maximum value: " + datapoint.maximum());
                }
            } else {
                System.out.println("The returned data list is empty");
            }

        } catch (CloudWatchException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }
```

- API の詳細については、「AWS SDK for Java 2.x API リファレンス」の「GetMetricStatistics」を参照してください。

Kotlin

SDK for Kotlin

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
suspend fun getAndDisplayMetricStatistics(nameSpaceVal: String, metVal: String,
 metricOption: String, date: String, myDimension: Dimension) {
    val start = Instant.parse(date)
    val endDate = Instant.now()
```

```
    val statisticsRequest = GetMetricStatisticsRequest {
        endTime = aws.smithy.kotlin.runtime.time.Instant(endDate)
        startTime = aws.smithy.kotlin.runtime.time.Instant(start)
        dimensions = listOf(myDimension)
        metricName = metVal
        namespace = nameSpaceVal
        period = 86400
        statistics = listOf(Statistic.fromValue(metricOption))
    }

    CloudWatchClient { region = "us-east-1" }.use { cwClient ->
        val response = cwClient.getMetricStatistics(statisticsRequest)
        val data = response.datapoints
        if (data != null) {
            if (data.isNotEmpty()) {
                for (datapoint in data) {
                    println("Timestamp: ${datapoint.timestamp} Maximum value:
 ${datapoint.maximum}")
                }
            } else {
                println("The returned data list is empty")
            }
        }
    }
}
```

- API の詳細については、「AWS SDK for Kotlin API リファレンス」の「GetMetricStatistics」を参照してください。

Python

SDK for Python (Boto3)

> **Note**
>
> GitHub には、その他のリソースもあります。完全な例を見つけて、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
class CloudWatchWrapper:
```

```
    """Encapsulates Amazon CloudWatch functions."""

    def __init__(self, cloudwatch_resource):
        """
        :param cloudwatch_resource: A Boto3 CloudWatch resource.
        """
        self.cloudwatch_resource = cloudwatch_resource


    def get_metric_statistics(self, namespace, name, start, end, period,
stat_types):
        """
        Gets statistics for a metric within a specified time span. Metrics are
grouped
        into the specified period.

        :param namespace: The namespace of the metric.
        :param name: The name of the metric.
        :param start: The UTC start time of the time span to retrieve.
        :param end: The UTC end time of the time span to retrieve.
        :param period: The period, in seconds, in which to group metrics. The
period
                       must match the granularity of the metric, which depends on
                       the metric's age. For example, metrics that are older than
                       three hours have a one-minute granularity, so the period
must
                       be at least 60 and must be a multiple of 60.
        :param stat_types: The type of statistics to retrieve, such as average
value
                           or maximum value.
        :return: The retrieved statistics for the metric.
        """
        try:
            metric = self.cloudwatch_resource.Metric(namespace, name)
            stats = metric.get_statistics(
                StartTime=start, EndTime=end, Period=period,
Statistics=stat_types
            )
            logger.info(
                "Got %s statistics for %s.", len(stats["Datapoints"]),
stats["Label"]
            )
        except ClientError:
```

```
            logger.exception("Couldn't get statistics for %s.%s.", namespace,
name)
            raise
        else:
            return stats
```

- API の詳細については、AWS SDK for Python (Boto3) API リファレンスの「GetMetricStatistics」を参照してください。

AWS SDK デベロッパーガイドとコード例の完全なリストについては、「AWS SDK で CloudWatch を使用する」を参照してください。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK または CLI で **GetMetricWidgetImage** を使用する

以下のコード例は、GetMetricWidgetImage の使用方法を示しています。

アクション例は、より大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。次のコード例で、このアクションのコンテキストを確認できます。

- メトリクス、ダッシュボード、およびアラームの使用を開始する

.NET

AWS SDK for .NET

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Get an image for a metric graphed over time.
    /// </summary>
    /// <param name="metricNamespace">The namespace of the metric.</param>
    /// <param name="metric">The name of the metric.</param>
    /// <param name="stat">The name of the stat to chart.</param>
```

```
    /// <param name="period">The period to use for the chart.</param>
    /// <returns>A memory stream for the chart image.</returns>
    public async Task<MemoryStream> GetTimeSeriesMetricImage(string
metricNamespace, string metric, string stat, int period)
    {
        var metricImageWidget = new
        {
            title = "Example Metric Graph",
            view = "timeSeries",
            stacked = false,
            period = period,
            width = 1400,
            height = 600,
            metrics = new List<List<object>>
                { new() { metricNamespace, metric, new { stat } } }
        };

        var metricImageWidgetString =
JsonSerializer.Serialize(metricImageWidget);
        var imageResponse = await _amazonCloudWatch.GetMetricWidgetImageAsync(
            new GetMetricWidgetImageRequest()
            {
                MetricWidget = metricImageWidgetString
            });

        return imageResponse.MetricWidgetImage;
    }

    /// <summary>
    /// Save a metric image to a file.
    /// </summary>
    /// <param name="memoryStream">The MemoryStream for the metric image.</param>
    /// <param name="metricName">The name of the metric.</param>
    /// <returns>The path to the file.</returns>
    public string SaveMetricImage(MemoryStream memoryStream, string metricName)
    {
        var metricFileName = $"{metricName}_{DateTime.Now.Ticks}.png";
        using var sr = new StreamReader(memoryStream);
        // Writes the memory stream to a file.
        File.WriteAllBytes(metricFileName, memoryStream.ToArray());
        var filePath = Path.Join(AppDomain.CurrentDomain.BaseDirectory,
            metricFileName);
        return filePath;
    }
```

- API の詳細については、「AWS SDK for .NET API リファレンス」の「GetMetricWidgetImage」を参照してください。

Java

SDK for Java 2.x

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
   public static void getAndOpenMetricImage(CloudWatchClient cw, String
fileName) {
       System.out.println("Getting Image data for custom metric.");
       try {
           String myJSON = "{\n" +
                   "  \"title\": \"Example Metric Graph\",\n" +
                   "  \"view\": \"timeSeries\",\n" +
                   "  \"stacked \": false,\n" +
                   "  \"period\": 10,\n" +
                   "  \"width\": 1400,\n" +
                   "  \"height\": 600,\n" +
                   "  \"metrics\": [\n" +
                   "    [\n" +
                   "      \"AWS/Billing\",\n" +
                   "      \"EstimatedCharges\",\n" +
                   "      \"Currency\",\n" +
                   "      \"USD\"\n" +
                   "    ]\n" +
                   "  ]\n" +
                   "}";

           GetMetricWidgetImageRequest imageRequest =
GetMetricWidgetImageRequest.builder()
                   .metricWidget(myJSON)
                   .build();
```

```
            GetMetricWidgetImageResponse response =
 cw.getMetricWidgetImage(imageRequest);
            SdkBytes sdkBytes = response.metricWidgetImage();
            byte[] bytes = sdkBytes.asByteArray();
            File outputFile = new File(fileName);
            try (FileOutputStream outputStream = new
 FileOutputStream(outputFile)) {
                outputStream.write(bytes);
            }

        } catch (CloudWatchException | IOException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }
```

- API の詳細については、「AWS SDK for Java 2.x API リファレンス」の「GetMetricWidgetImage」を参照してください。

Kotlin

SDK for Kotlin

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
suspend fun getAndOpenMetricImage(fileName: String) {
    println("Getting Image data for custom metric.")
    val myJSON = """{
        "title": "Example Metric Graph",
        "view": "timeSeries",
        "stacked ": false,
        "period": 10,
        "width": 1400,
        "height": 600,
        "metrics": [
            [
```

```
            "AWS/Billing",
            "EstimatedCharges",
            "Currency",
            "USD"
            ]
        ]
        }"""

    val imageRequest = GetMetricWidgetImageRequest {
        metricWidget = myJSON
    }

    CloudWatchClient { region = "us-east-1" }.use { cwClient ->
        val response = cwClient.getMetricWidgetImage(imageRequest)
        val bytes = response.metricWidgetImage
        if (bytes != null) {
            File(fileName).writeBytes(bytes)
        }
    }
    println("You have successfully written data to $fileName")
}
```

- API の詳細については、「AWS SDK for Kotlin API リファレンス」の「GetMetricWidgetImage」を参照してください。

AWS SDK デベロッパーガイドとコード例の完全なリストについては、「AWS SDK で CloudWatch を使用する」を参照してください。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK または CLI で **ListDashboards** を使用する

以下のコード例は、ListDashboards の使用方法を示しています。

.NET

AWS SDK for .NET

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
/// <summary>
/// Get a list of dashboards.
/// </summary>
/// <returns>A list of DashboardEntry objects.</returns>
public async Task<List<DashboardEntry>> ListDashboards()
{
    var results = new List<DashboardEntry>();
    var paginateDashboards = _amazonCloudWatch.Paginators.ListDashboards(
        new ListDashboardsRequest());
    // Get the entire list using the paginator.
    await foreach (var data in paginateDashboards.DashboardEntries)
    {
        results.Add(data);
    }

    return results;
}
```

- API の詳細については、「AWS SDK for .NET API リファレンス」の「ListDashboards」を参照してください。

Java

SDK for Java 2.x

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
    public static void listDashboards(CloudWatchClient cw) {
        try {
            ListDashboardsIterable listRes = cw.listDashboardsPaginator();
            listRes.stream()
                    .flatMap(r -> r.dashboardEntries().stream())
                    .forEach(entry -> {
                        System.out.println("Dashboard name is: " +
entry.dashboardName());
                        System.out.println("Dashboard ARN is: " +
entry.dashboardArn());
                    });

        } catch (CloudWatchException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }
```

- API の詳細については、「AWS SDK for Java 2.x API リファレンス」の「ListDashboards」を参照してください。

Kotlin

SDK for Kotlin

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
suspend fun listDashboards() {
    CloudWatchClient { region = "us-east-1" }.use { cwClient ->
        cwClient.listDashboardsPaginated({})
            .transform { it.dashboardEntries?.forEach { obj -> emit(obj) } }
            .collect { obj ->
                println("Name is ${obj.dashboardName}")
                println("Dashboard ARN is ${obj.dashboardArn}")
            }
    }
```

```
}
```

- API の詳細については、「AWS SDK for Kotlin API リファレンス」の「ListDashboards」を参照してください。

PowerShell

Tools for PowerShell

例 1: アカウントのダッシュボードのコレクションを返します。

```
Get-CWDashboardList
```

出力:

```
DashboardArn DashboardName LastModified        Size
------------ ------------- ------------        ----
arn:...      Dashboard1    7/6/2017 8:14:15 PM 252
```

例 2: 名前が 'dev' で始まるアカウントのダッシュボードのコレクションを返します。

```
Get-CWDashboardList -DashboardNamePrefix dev
```

- API の詳細については、「AWS Tools for PowerShell コマンドレットリファレンス」の「ListDashboards」を参照してください。

AWS SDK デベロッパーガイドとコード例の完全なリストについては、「AWS SDK で CloudWatch を使用する」を参照してください。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK または CLI で **ListMetrics** を使用する

以下のコード例は、ListMetrics の使用方法を示しています。

アクション例は、より大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。次のコード例で、このアクションのコンテキストを確認できます。

- メトリクス、ダッシュボード、およびアラームの使用を開始する

- メトリクスとアラームを管理する

.NET

AWS SDK for .NET

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// List metrics available, optionally within a namespace.
    /// </summary>
    /// <param name="metricNamespace">Optional CloudWatch namespace to use when
listing metrics.</param>
    /// <param name="filter">Optional dimension filter.</param>
    /// <param name="metricName">Optional metric name filter.</param>
    /// <returns>The list of metrics.</returns>
    public async Task<List<Metric>> ListMetrics(string? metricNamespace = null,
DimensionFilter? filter = null, string? metricName = null)
    {
        var results = new List<Metric>();
        var paginateMetrics = _amazonCloudWatch.Paginators.ListMetrics(
            new ListMetricsRequest
            {
                Namespace = metricNamespace,
                Dimensions = filter != null ? new List<DimensionFilter>
{ filter } : null,
                MetricName = metricName
            });
        // Get the entire list using the paginator.
        await foreach (var metric in paginateMetrics.Metrics)
        {
            results.Add(metric);
        }

        return results;
    }
```

- API の詳細については、「AWS SDK for .NET API リファレンス」の「ListMetrics」を参照してください。

C++

SDK for C++

> **Note**
>
> GitHub には、その他のリソースもあります。完全な例を見つけて、AWS コード例リポジトリでの設定と実行の方法を確認してください。

必要なファイルを含めます。

```
#include <aws/core/Aws.h>
#include <aws/monitoring/CloudWatchClient.h>
#include <aws/monitoring/model/ListMetricsRequest.h>
#include <aws/monitoring/model/ListMetricsResult.h>
#include <iomanip>
#include <iostream>
```

メトリクスを一覧表示します。

```
        Aws::CloudWatch::CloudWatchClient cw;
        Aws::CloudWatch::Model::ListMetricsRequest request;

        if (argc > 1)
        {
            request.SetMetricName(argv[1]);
        }

        if (argc > 2)
        {
            request.SetNamespace(argv[2]);
        }

        bool done = false;
        bool header = false;
        while (!done)
```

```
        {
            auto outcome = cw.ListMetrics(request);
            if (!outcome.IsSuccess())
            {
                std::cout << "Failed to list CloudWatch metrics:" <<
                    outcome.GetError().GetMessage() << std::endl;
                break;
            }

            if (!header)
            {
                std::cout << std::left << std::setw(48) << "MetricName" <<
                    std::setw(32) << "Namespace" << "DimensionNameValuePairs" <<
                    std::endl;
                header = true;
            }

            const auto &metrics = outcome.GetResult().GetMetrics();
            for (const auto &metric : metrics)
            {
                std::cout << std::left << std::setw(48) <<
                    metric.GetMetricName() << std::setw(32) <<
                    metric.GetNamespace();
                const auto &dimensions = metric.GetDimensions();
                for (auto iter = dimensions.cbegin();
                    iter != dimensions.cend(); ++iter)
                {
                    const auto &dimkv = *iter;
                    std::cout << dimkv.GetName() << " = " << dimkv.GetValue();
                    if (iter + 1 != dimensions.cend())
                    {
                        std::cout << ", ";
                    }
                }
                std::cout << std::endl;
            }

            const auto &next_token = outcome.GetResult().GetNextToken();
            request.SetNextToken(next_token);
            done = next_token.empty();
        }
```

- API の詳細については、「AWS SDK for C++ API リファレンス」の「ListMetrics」を参照してください。

CLI

AWS CLI

Amazon SNS のメトリクスを一覧表示するには

次の `list-metrics` の例は、Amazon SNS のメトリクスを表示します。

```
aws cloudwatch list-metrics \
    --namespace "AWS/SNS"
```

出力:

```
{
    "Metrics": [
        {
            "Namespace": "AWS/SNS",
            "Dimensions": [
                {
                    "Name": "TopicName",
                    "Value": "NotifyMe"
                }
            ],
            "MetricName": "PublishSize"
        },
        {
            "Namespace": "AWS/SNS",
            "Dimensions": [
                {
                    "Name": "TopicName",
                    "Value": "CFO"
                }
            ],
            "MetricName": "PublishSize"
        },
        {
            "Namespace": "AWS/SNS",
            "Dimensions": [
                {
```

```
                    "Name": "TopicName",
                    "Value": "NotifyMe"
                }
            ],
            "MetricName": "NumberOfNotificationsFailed"
        },
        {
            "Namespace": "AWS/SNS",
            "Dimensions": [
                {
                    "Name": "TopicName",
                    "Value": "NotifyMe"
                }
            ],
            "MetricName": "NumberOfNotificationsDelivered"
        },
        {
            "Namespace": "AWS/SNS",
            "Dimensions": [
                {
                    "Name": "TopicName",
                    "Value": "NotifyMe"
                }
            ],
            "MetricName": "NumberOfMessagesPublished"
        },
        {
            "Namespace": "AWS/SNS",
            "Dimensions": [
                {
                    "Name": "TopicName",
                    "Value": "CFO"
                }
            ],
            "MetricName": "NumberOfMessagesPublished"
        },
        {
            "Namespace": "AWS/SNS",
            "Dimensions": [
                {
                    "Name": "TopicName",
                    "Value": "CFO"
                }
            ],
```

```
            "MetricName": "NumberOfNotificationsDelivered"
        },
        {
            "Namespace": "AWS/SNS",
            "Dimensions": [
                {
                    "Name": "TopicName",
                    "Value": "CFO"
                }
            ],
            "MetricName": "NumberOfNotificationsFailed"
        }
    ]
}
```

- API の詳細については、AWS CLI コマンドリファレンスの「ListMetrics」を参照してください。

Java

SDK for Java 2.x

> **Note**
>
> GitHub には、その他のリソースもあります。完全な例を見つけて、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.cloudwatch.CloudWatchClient;
import software.amazon.awssdk.services.cloudwatch.model.CloudWatchException;
import software.amazon.awssdk.services.cloudwatch.model.ListMetricsRequest;
import software.amazon.awssdk.services.cloudwatch.model.ListMetricsResponse;
import software.amazon.awssdk.services.cloudwatch.model.Metric;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
```

```
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class ListMetrics {
    public static void main(String[] args) {
        final String usage = """

                Usage:
                  <namespace>\s

                Where:
                  namespace - The namespace to filter against (for example, AWS/
EC2).\s
                """;

        if (args.length != 1) {
            System.out.println(usage);
            System.exit(1);
        }

        String namespace = args[0];
        Region region = Region.US_EAST_1;
        CloudWatchClient cw = CloudWatchClient.builder()
                .region(region)
                .build();

        listMets(cw, namespace);
        cw.close();
    }

    public static void listMets(CloudWatchClient cw, String namespace) {
        boolean done = false;
        String nextToken = null;

        try {
            while (!done) {

                ListMetricsResponse response;
                if (nextToken == null) {
                    ListMetricsRequest request = ListMetricsRequest.builder()
                            .namespace(namespace)
                            .build();

                    response = cw.listMetrics(request);
```

```
                } else {
                    ListMetricsRequest request = ListMetricsRequest.builder()
                            .namespace(namespace)
                            .nextToken(nextToken)
                            .build();

                    response = cw.listMetrics(request);
                }

                for (Metric metric : response.metrics()) {
                    System.out.printf("Retrieved metric %s",
 metric.metricName());
                    System.out.println();
                }

                if (response.nextToken() == null) {
                    done = true;
                } else {
                    nextToken = response.nextToken();
                }
            }

        } catch (CloudWatchException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }
}
```

- API の詳細については、「AWS SDK for Java 2.x API リファレンス」の「ListMetrics」を参照してください。

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

SDK モジュールとクライアントモジュールをインポートし、API を呼び出します。

```
import { ListMetricsCommand } from "@aws-sdk/client-cloudwatch";
import { client } from "../libs/client.js";

export const main = () => {
  // Use the AWS console to see available namespaces and metric names. Custom
 metrics can also be created.
  // https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/
viewing_metrics_with_cloudwatch.html
  const command = new ListMetricsCommand({
    Dimensions: [
      {
        Name: "LogGroupName",
      },
    ],
    MetricName: "IncomingLogEvents",
    Namespace: "AWS/Logs",
  });

  return client.send(command);
};
```

別のモジュールでクライアントを作成し、エクスポートします。

```
import { CloudWatchClient } from "@aws-sdk/client-cloudwatch";

export const client = new CloudWatchClient({});
```

- 詳細については、「AWS SDK for JavaScript デベロッパーガイド」を参照してください。
- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「ListMetrics」を参照してください。

SDK for JavaScript (v2)

> **Note**
>
> GitHub には、その他のリソースもあります。完全な例を見つけて、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create CloudWatch service object
var cw = new AWS.CloudWatch({ apiVersion: "2010-08-01" });

var params = {
  Dimensions: [
    {
      Name: "LogGroupName" /* required */,
    },
  ],
  MetricName: "IncomingLogEvents",
  Namespace: "AWS/Logs",
};

cw.listMetrics(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Metrics", JSON.stringify(data.Metrics));
  }
});
```

- 詳細については、「AWS SDK for JavaScript デベロッパーガイド」を参照してください。
- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「ListMetrics」を参照してください。

Kotlin

SDK for Kotlin

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
suspend fun listMets(namespaceVal: String?): ArrayList<String>? {
    val metList = ArrayList<String>()
    val request = ListMetricsRequest {
        namespace = namespaceVal
    }
    CloudWatchClient { region = "us-east-1" }.use { cwClient ->
        val reponse = cwClient.listMetrics(request)
        reponse.metrics?.forEach { metrics ->
            val data = metrics.metricName
            if (!metList.contains(data)) {
                metList.add(data!!)
            }
        }
    }
    return metList
}
```

- API の詳細については、「AWS SDK for Kotlin API リファレンス」の「ListMetrics」を参照してください。

Python

SDK for Python (Boto3)

> **Note**
>
> GitHub には、その他のリソースもあります。完全な例を見つけて、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
class CloudWatchWrapper:
    """Encapsulates Amazon CloudWatch functions."""

    def __init__(self, cloudwatch_resource):
        """
        :param cloudwatch_resource: A Boto3 CloudWatch resource.
        """
        self.cloudwatch_resource = cloudwatch_resource
```

```
    def list_metrics(self, namespace, name, recent=False):
        """
        Gets the metrics within a namespace that have the specified name.
        If the metric has no dimensions, a single metric is returned.
        Otherwise, metrics for all dimensions are returned.

        :param namespace: The namespace of the metric.
        :param name: The name of the metric.
        :param recent: When True, only metrics that have been active in the last
                       three hours are returned.
        :return: An iterator that yields the retrieved metrics.
        """
        try:
            kwargs = {"Namespace": namespace, "MetricName": name}
            if recent:
                kwargs["RecentlyActive"] = "PT3H"  # List past 3 hours only
            metric_iter = self.cloudwatch_resource.metrics.filter(**kwargs)
            logger.info("Got metrics for %s.%s.", namespace, name)
        except ClientError:
            logger.exception("Couldn't get metrics for %s.%s.", namespace, name)
            raise
        else:
            return metric_iter
```

- API の詳細については、AWS SDK for Python (Boto3) API リファレンスの「ListMetrics」を参照してください。

Ruby

SDK for Ruby

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
# Lists available metrics for a metric namespace in Amazon CloudWatch.
#
```

```
# @param cloudwatch_client [Aws::CloudWatch::Client]
#   An initialized CloudWatch client.
# @param metric_namespace [String] The namespace of the metric.
# @example
#   list_metrics_for_namespace(
#     Aws::CloudWatch::Client.new(region: 'us-east-1'),
#     'SITE/TRAFFIC'
#   )
def list_metrics_for_namespace(cloudwatch_client, metric_namespace)
  response = cloudwatch_client.list_metrics(namespace: metric_namespace)

  if response.metrics.count.positive?
    response.metrics.each do |metric|
      puts "  Metric name: #{metric.metric_name}"
      if metric.dimensions.count.positive?
        puts "    Dimensions:"
        metric.dimensions.each do |dimension|
          puts "      Name: #{dimension.name}, Value: #{dimension.value}"
        end
      else
        puts "No dimensions found."
      end
    end
  else
    puts "No metrics found for namespace '#{metric_namespace}'. " \
      "Note that it could take up to 15 minutes for recently-added metrics " \
      "to become available."
  end
end

# Example usage:
def run_me
  metric_namespace = "SITE/TRAFFIC"
  # Replace us-west-2 with the AWS Region you're using for Amazon CloudWatch.
  region = "us-east-1"

  cloudwatch_client = Aws::CloudWatch::Client.new(region: region)

  # Add three datapoints.
  puts "Continuing..." unless datapoint_added_to_metric?(
    cloudwatch_client,
    metric_namespace,
    "UniqueVisitors",
    "SiteName",
```

```
    "example.com",
    5_885.0,
    "Count"
  )

  puts "Continuing..." unless datapoint_added_to_metric?(
    cloudwatch_client,
    metric_namespace,
    "UniqueVisits",
    "SiteName",
    "example.com",
    8_628.0,
    "Count"
  )

  puts "Continuing..." unless datapoint_added_to_metric?(
    cloudwatch_client,
    metric_namespace,
    "PageViews",
    "PageURL",
    "example.html",
    18_057.0,
    "Count"
  )

  puts "Metrics for namespace '#{metric_namespace}':"
  list_metrics_for_namespace(cloudwatch_client, metric_namespace)
end

run_me if $PROGRAM_NAME == __FILE__
```

- API の詳細については、「AWS SDK for Ruby API リファレンス」の「ListMetrics」を参照してください。

SAP ABAP

SDK for SAP ABAP

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
   "The following list-metrics example displays the metrics for Amazon
CloudWatch."
   TRY.
       oo_result = lo_cwt->listmetrics(            " oo_result is returned for
testing purposes. "
         iv_namespace = iv_namespace
       ).
       DATA(lt_metrics) = oo_result->get_metrics( ).
       MESSAGE 'Metrics retrieved.' TYPE 'I'.
     CATCH /aws1/cx_cwtinvparamvalueex .
       MESSAGE 'The specified argument was not valid.' TYPE 'E'.
   ENDTRY.
```

- API の詳細については、「AWS SDK for SAP ABAP API リファレンス」の「ListMetrics」を参照してください。

AWS SDK デベロッパーガイドとコード例の完全なリストについては、「AWS SDK で CloudWatch を使用する」を参照してください。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK または CLI で **PutAnomalyDetector** を使用する

以下のコード例は、PutAnomalyDetector の使用方法を示しています。

アクション例は、より大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。次のコード例で、このアクションのコンテキストを確認できます。

- メトリクス、ダッシュボード、およびアラームの使用を開始する

.NET

AWS SDK for .NET

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Add an anomaly detector for a single metric.
    /// </summary>
    /// <param name="anomalyDetector">A single metric anomaly detector.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> PutAnomalyDetector(SingleMetricAnomalyDetector
anomalyDetector)
    {
        var putAlarmDetectorResult = await
_amazonCloudWatch.PutAnomalyDetectorAsync(
            new PutAnomalyDetectorRequest()
            {
                SingleMetricAnomalyDetector = anomalyDetector
            });

        return putAlarmDetectorResult.HttpStatusCode == HttpStatusCode.OK;
    }
```

- API の詳細については、「AWS SDK for .NET API Reference」(API リファレンス) の「PutAnomalyDetector」を参照してください。

Java

SDK for Java 2.x

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
   public static void addAnomalyDetector(CloudWatchClient cw, String fileName) {
       try {
           // Read values from the JSON file.
           JsonParser parser = new JsonFactory().createParser(new
File(fileName));
           com.fasterxml.jackson.databind.JsonNode rootNode = new
ObjectMapper().readTree(parser);
           String customMetricNamespace =
rootNode.findValue("customMetricNamespace").asText();
           String customMetricName =
rootNode.findValue("customMetricName").asText();

           SingleMetricAnomalyDetector singleMetricAnomalyDetector =
SingleMetricAnomalyDetector.builder()
                   .metricName(customMetricName)
                   .namespace(customMetricNamespace)
                   .stat("Maximum")
                   .build();

           PutAnomalyDetectorRequest anomalyDetectorRequest =
PutAnomalyDetectorRequest.builder()
                   .singleMetricAnomalyDetector(singleMetricAnomalyDetector)
                   .build();

           cw.putAnomalyDetector(anomalyDetectorRequest);
           System.out.println("Added anomaly detector for metric " +
customMetricName + ".");

       } catch (CloudWatchException | IOException e) {
           System.err.println(e.getMessage());
           System.exit(1);
       }
   }
```

- API の詳細については、「AWS SDK for Java 2.x API Reference」(API リファレンス) の「PutAnomalyDetector」を参照してください。

Kotlin

SDK for Kotlin

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
suspend fun addAnomalyDetector(fileName: String?) {
    // Read values from the JSON file.
    val parser = JsonFactory().createParser(File(fileName))
    val rootNode = ObjectMapper().readTree<JsonNode>(parser)
    val customMetricNamespace =
 rootNode.findValue("customMetricNamespace").asText()
    val customMetricName = rootNode.findValue("customMetricName").asText()

    val singleMetricAnomalyDetectorVal = SingleMetricAnomalyDetector {
        metricName = customMetricName
        namespace = customMetricNamespace
        stat = "Maximum"
    }

    val anomalyDetectorRequest = PutAnomalyDetectorRequest {
        singleMetricAnomalyDetector = singleMetricAnomalyDetectorVal
    }

    CloudWatchClient { region = "us-east-1" }.use { cwClient ->
        cwClient.putAnomalyDetector(anomalyDetectorRequest)
        println("Added anomaly detector for metric $customMetricName.")
    }
}
```

- API の詳細については、「AWS SDK for Kotlin API reference」(SDK for Kotlin API リファレンス) の「PutAnomalyDetector」を参照してください。

AWS SDK デベロッパーガイドとコード例の完全なリストについては、「AWS SDK で CloudWatch を使用する」を参照してください。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK または CLI で **PutDashboard** を使用する

以下のコード例は、PutDashboard の使用方法を示しています。

アクション例は、より大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。次のコード例で、このアクションのコンテキストを確認できます。

- メトリクス、ダッシュボード、およびアラームの使用を開始する

.NET

AWS SDK for .NET

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Set up a dashboard using a call to the wrapper class.
    /// </summary>
    /// <param name="customMetricNamespace">The metric namespace.</param>
    /// <param name="customMetricName">The metric name.</param>
    /// <param name="dashboardName">The name of the dashboard.</param>
    /// <returns>A list of validation messages.</returns>
    private static async Task<List<DashboardValidationMessage>> SetupDashboard(
        string customMetricNamespace, string customMetricName, string
dashboardName)
    {
        // Get the dashboard model from configuration.
        var newDashboard = new DashboardModel();
        _configuration.GetSection("dashboardExampleBody").Bind(newDashboard);

        // Add a new metric to the dashboard.
        newDashboard.Widgets.Add(new Widget
        {
            Height = 8,
            Width = 8,
            Y = 8,
```

```
            X = 0,
            Type = "metric",
            Properties = new Properties
            {
                Metrics = new List<List<object>>
                    { new() { customMetricNamespace, customMetricName } },
                View = "timeSeries",
                Region = "us-east-1",
                Stat = "Sum",
                Period = 86400,
                YAxis = new YAxis { Left = new Left { Min = 0, Max = 100 } },
                Title = "Custom Metric Widget",
                LiveData = true,
                Sparkline = true,
                Trend = true,
                Stacked = false,
                SetPeriodToTimeRange = false
            }
        });

        var newDashboardString = JsonSerializer.Serialize(newDashboard,
            new JsonSerializerOptions
            { DefaultIgnoreCondition = JsonIgnoreCondition.WhenWritingNull });
        var validationMessages =
            await _cloudWatchWrapper.PutDashboard(dashboardName,
 newDashboardString);

        return validationMessages;
    }

    /// <summary>
    /// Wrapper to create or add to a dashboard with metrics.
    /// </summary>
    /// <param name="dashboardName">The name for the dashboard.</param>
    /// <param name="dashboardBody">The metric data in JSON for the dashboard.</
param>
    /// <returns>A list of validation messages for the dashboard.</returns>
    public async Task<List<DashboardValidationMessage>> PutDashboard(string
 dashboardName,
        string dashboardBody)
    {
        // Updating a dashboard replaces all contents.
        // Best practice is to include a text widget indicating this dashboard
 was created programmatically.
```

```
        var dashboardResponse = await _amazonCloudWatch.PutDashboardAsync(
            new PutDashboardRequest()
            {
                DashboardName = dashboardName,
                DashboardBody = dashboardBody
            });

        return dashboardResponse.DashboardValidationMessages;
    }
```

- API の詳細については、「AWS SDK for .NET API リファレンス」の「PutDashboard」を参照してください。

Java

SDK for Java 2.x

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
   public static void createDashboardWithMetrics(CloudWatchClient cw, String
dashboardName, String fileName) {
        try {
            PutDashboardRequest dashboardRequest = PutDashboardRequest.builder()
                    .dashboardName(dashboardName)
                    .dashboardBody(readFileAsString(fileName))
                    .build();

            PutDashboardResponse response = cw.putDashboard(dashboardRequest);
            System.out.println(dashboardName + " was successfully created.");
            List<DashboardValidationMessage> messages =
response.dashboardValidationMessages();
            if (messages.isEmpty()) {
                System.out.println("There are no messages in the new Dashboard");
            } else {
                for (DashboardValidationMessage message : messages) {
```

```
                    System.out.println("Message is: " + message.message());
                }
            }

        } catch (CloudWatchException | IOException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }
```

- API の詳細については、「AWS SDK for Java 2.x API リファレンス」の「PutDashboard」を参照してください。

Kotlin

SDK for Kotlin

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
suspend fun createDashboardWithMetrics(dashboardNameVal: String, fileNameVal:
 String) {
    val dashboardRequest = PutDashboardRequest {
        dashboardName = dashboardNameVal
        dashboardBody = readFileAsString(fileNameVal)
    }

    CloudWatchClient { region = "us-east-1" }.use { cwClient ->
        val response = cwClient.putDashboard(dashboardRequest)
        println("$dashboardNameVal was successfully created.")
        val messages = response.dashboardValidationMessages
        if (messages != null) {
            if (messages.isEmpty()) {
                println("There are no messages in the new Dashboard")
            } else {
                for (message in messages) {
                    println("Message is: ${message.message}")
```

```
                }
            }
        }
    }
}
```

- API の詳細については、「AWS SDK for Kotlin API リファレンス」の「PutDashboard」を参照してください。

PowerShell

Tools for PowerShell

例 1: 'Dashboard1' という名前のダッシュボードを作成または更新して、2 つのメトリクスウィジェットを並べて表示します。

```
$dashBody = @"
{
    "widgets":[
        {
             "type":"metric",
             "x":0,
             "y":0,
             "width":12,
             "height":6,
             "properties":{
                "metrics":[
                   [
                      "AWS/EC2",
                      "CPUUtilization",
                      "InstanceId",
                      "i-012345"
                   ]
                ],
                "period":300,
                "stat":"Average",
                "region":"us-east-1",
                "title":"EC2 Instance CPU"
             }
        },
        {
```

```
            "type":"metric",
            "x":12,
            "y":0,
            "width":12,
            "height":6,
            "properties":{
               "metrics":[
                  [
                     "AWS/S3",
                     "BucketSizeBytes",
                     "BucketName",
                     "MyBucketName"
                  ]
               ],
               "period":86400,
               "stat":"Maximum",
               "region":"us-east-1",
               "title":"MyBucketName bytes"
            }
         }
    ]
}
"@

Write-CWDashboard -DashboardName Dashboard1 -DashboardBody $dashBody
```

例 2: ダッシュボードを作成または更新し、ダッシュボードを説明するコンテンツをコマンドレットにパイプします。

```
$dashBody = @"
{
...
}
"@

$dashBody | Write-CWDashboard -DashboardName Dashboard1
```

- API の詳細については、「AWS Tools for PowerShell コマンドレットリファレンス」の「PutDashboard」を参照してください。

AWS SDK デベロッパーガイドとコード例の完全なリストについては、「AWS SDK で CloudWatch を使用する」を参照してください。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK または CLI で **PutMetricAlarm** を使用する

以下のコード例は、PutMetricAlarm の使用方法を示しています。

アクション例は、より大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。次のコード例で、このアクションのコンテキストを確認できます。

- アラームの使用を開始
- メトリクス、ダッシュボード、およびアラームの使用を開始する
- メトリクスとアラームを管理する

.NET

AWS SDK for .NET

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Add a metric alarm to send an email when the metric passes a threshold.
    /// </summary>
    /// <param name="alarmDescription">A description of the alarm.</param>
    /// <param name="alarmName">The name for the alarm.</param>
    /// <param name="comparison">The type of comparison to use.</param>
    /// <param name="metricName">The name of the metric for the alarm.</param>
    /// <param name="metricNamespace">The namespace of the metric.</param>
    /// <param name="threshold">The threshold value for the alarm.</param>
    /// <param name="alarmActions">Optional actions to execute when in an alarm
state.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> PutMetricEmailAlarm(string alarmDescription, string
alarmName, ComparisonOperator comparison,
```

```
        string metricName, string metricNamespace, double threshold, List<string>
alarmActions = null!)
    {
        try
        {
            var putEmailAlarmResponse = await
_amazonCloudWatch.PutMetricAlarmAsync(
                new PutMetricAlarmRequest()
                {
                    AlarmActions = alarmActions,
                    AlarmDescription = alarmDescription,
                    AlarmName = alarmName,
                    ComparisonOperator = comparison,
                    Threshold = threshold,
                    Namespace = metricNamespace,
                    MetricName = metricName,
                    EvaluationPeriods = 1,
                    Period = 10,
                    Statistic = new Statistic("Maximum"),
                    DatapointsToAlarm = 1,
                    TreatMissingData = "ignore"
                });
            return putEmailAlarmResponse.HttpStatusCode == HttpStatusCode.OK;
        }
        catch (LimitExceededException lex)
        {
            _logger.LogError(lex, $"Unable to add alarm {alarmName}. Alarm quota
has already been reached.");
        }

        return false;
    }

    /// <summary>
    /// Add specific email actions to a list of action strings for a CloudWatch
alarm.
    /// </summary>
    /// <param name="accountId">The AccountId for the alarm.</param>
    /// <param name="region">The region for the alarm.</param>
    /// <param name="emailTopicName">An Amazon Simple Notification Service (SNS)
topic for the alarm email.</param>
    /// <param name="alarmActions">Optional list of existing alarm actions to
append to.</param>
    /// <returns>A list of string actions for an alarm.</returns>
```

```
    public List<string> AddEmailAlarmAction(string accountId, string region,
        string emailTopicName, List<string>? alarmActions = null)
    {
        alarmActions ??= new List<string>();
        var snsAlarmAction = $"arn:aws:sns:{region}:{accountId}:
{emailTopicName}";
        alarmActions.Add(snsAlarmAction);
        return alarmActions;
    }
```

- API の詳細については、「AWS SDK for .NET API リファレンス」の「PutMetricAlarm」を参照してください。

C++

SDK for C++

> **Note**
>
> GitHub には、その他のリソースもあります。完全な例を見つけて、AWS コード例リポジトリでの設定と実行の方法を確認してください。

必要なファイルを含めます。

```
#include <aws/core/Aws.h>
#include <aws/monitoring/CloudWatchClient.h>
#include <aws/monitoring/model/PutMetricAlarmRequest.h>
#include <iostream>
```

メトリクスを監視するアラームを作成します。

```
        Aws::CloudWatch::CloudWatchClient cw;
        Aws::CloudWatch::Model::PutMetricAlarmRequest request;
        request.SetAlarmName(alarm_name);
        request.SetComparisonOperator(
            Aws::CloudWatch::Model::ComparisonOperator::GreaterThanThreshold);
        request.SetEvaluationPeriods(1);
        request.SetMetricName("CPUUtilization");
```

```
        request.SetNamespace("AWS/EC2");
        request.SetPeriod(60);
        request.SetStatistic(Aws::CloudWatch::Model::Statistic::Average);
        request.SetThreshold(70.0);
        request.SetActionsEnabled(false);
        request.SetAlarmDescription("Alarm when server CPU exceeds 70%");
        request.SetUnit(Aws::CloudWatch::Model::StandardUnit::Seconds);

        Aws::CloudWatch::Model::Dimension dimension;
        dimension.SetName("InstanceId");
        dimension.SetValue(instanceId);

        request.AddDimensions(dimension);

        auto outcome = cw.PutMetricAlarm(request);
        if (!outcome.IsSuccess())
        {
            std::cout << "Failed to create CloudWatch alarm:" <<
                outcome.GetError().GetMessage() << std::endl;
        }
        else
        {
            std::cout << "Successfully created CloudWatch alarm " << alarm_name
                << std::endl;
        }
```

- API の詳細については、「AWS SDK for C++ API リファレンス」の「PutMetricAlarm」を参照してください。

CLI

AWS CLI

CPU 使用率が 70% を超えたときに Amazon Simple Notification Service の E メールメッセージを送信するには

次の例では、put-metric-alarm コマンドを使用して、CPU 使用率が 70% を超えたときに、Amazon Simple Notification Service に E メールメッセージを送信します。

```
aws cloudwatch put-metric-alarm --alarm-name cpu-mon --alarm-description "Alarm
 when CPU exceeds 70 percent" --metric-name CPUUtilization --namespace AWS/
```

```
EC2 --statistic Average --period 300 --threshold 70 --comparison-operator
 GreaterThanThreshold  --dimensions "Name=InstanceId,Value=i-12345678" --
evaluation-periods 2 --alarm-actions arn:aws:sns:us-east-1:111122223333:MyTopic
 --unit Percent
```

正常に完了すると、このコマンドはプロンプトに戻ります。同じ名前のアラームが既に存在する場合は、新しいアラームで上書きされます。

複数のディメンションを指定するには

次の例は、複数のディメンションを指定する方法を示しています。各ディメンションは名前/値のペアとして指定され、名前と値の間にはカンマが入ります。複数のディメンションはスペースで区切ります。

```
aws cloudwatch put-metric-alarm --alarm-name "Default_Test_Alarm3" --alarm-
description "The default example alarm" --namespace "CW EXAMPLE METRICS" --
metric-name Default_Test --statistic Average --period 60 --evaluation-periods 3
 --threshold 50 --comparison-operator GreaterThanOrEqualToThreshold --dimensions
 Name=key1,Value=value1 Name=key2,Value=value2
```

- API の詳細については、AWS CLI コマンドリファレンスの「PutMetricAlarm」を参照してください。

Java

SDK for Java 2.x

> **Note**
>
> GitHub には、その他のリソースもあります。完全な例を見つけて、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
    public static String createAlarm(CloudWatchClient cw, String fileName) {
        try {
            // Read values from the JSON file.
            JsonParser parser = new JsonFactory().createParser(new
File(fileName));
            com.fasterxml.jackson.databind.JsonNode rootNode = new
ObjectMapper().readTree(parser);
```

```
            String customMetricNamespace =
rootNode.findValue("customMetricNamespace").asText();
            String customMetricName =
rootNode.findValue("customMetricName").asText();
            String alarmName = rootNode.findValue("exampleAlarmName").asText();
            String emailTopic = rootNode.findValue("emailTopic").asText();
            String accountId = rootNode.findValue("accountId").asText();
            String region = rootNode.findValue("region").asText();

            // Create a List for alarm actions.
            List<String> alarmActions = new ArrayList<>();
            alarmActions.add("arn:aws:sns:" + region + ":" + accountId + ":" +
emailTopic);
            PutMetricAlarmRequest alarmRequest = PutMetricAlarmRequest.builder()
                    .alarmActions(alarmActions)
                    .alarmDescription("Example metric alarm")
                    .alarmName(alarmName)

.comparisonOperator(ComparisonOperator.GREATER_THAN_OR_EQUAL_TO_THRESHOLD)
                    .threshold(100.00)
                    .metricName(customMetricName)
                    .namespace(customMetricNamespace)
                    .evaluationPeriods(1)
                    .period(10)
                    .statistic("Maximum")
                    .datapointsToAlarm(1)
                    .treatMissingData("ignore")
                    .build();

            cw.putMetricAlarm(alarmRequest);
            System.out.println(alarmName + " was successfully created!");
            return alarmName;

        } catch (CloudWatchException | IOException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
        return "";
    }
```

- API の詳細については、「AWS SDK for Java 2.x API リファレンス」の「PutMetricAlarm」を参照してください。

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

SDK モジュールとクライアントモジュールをインポートし、API を呼び出します。

```
import { PutMetricAlarmCommand } from "@aws-sdk/client-cloudwatch";
import { client } from "../libs/client.js";

const run = async () => {
  // This alarm triggers when CPUUtilization exceeds 70% for one minute.
  const command = new PutMetricAlarmCommand({
    AlarmName: process.env.CLOUDWATCH_ALARM_NAME, // Set the value of
 CLOUDWATCH_ALARM_NAME to the name of an existing alarm.
    ComparisonOperator: "GreaterThanThreshold",
    EvaluationPeriods: 1,
    MetricName: "CPUUtilization",
    Namespace: "AWS/EC2",
    Period: 60,
    Statistic: "Average",
    Threshold: 70.0,
    ActionsEnabled: false,
    AlarmDescription: "Alarm when server CPU exceeds 70%",
    Dimensions: [
      {
        Name: "InstanceId",
        Value: process.env.EC2_INSTANCE_ID, // Set the value of EC_INSTANCE_ID to
 the Id of an existing Amazon EC2 instance.
      },
    ],
    Unit: "Percent",
  });

  try {
    return await client.send(command);
  } catch (err) {
    console.error(err);
```

```
  }
};

export default run();
```

別のモジュールでクライアントを作成し、エクスポートします。

```
import { CloudWatchClient } from "@aws-sdk/client-cloudwatch";

export const client = new CloudWatchClient({});
```

- 詳細については、「AWS SDK for JavaScript デベロッパーガイド」を参照してください。
- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「PutMetricAlarm」を参照してください。

SDK for JavaScript (v2)

> **Note**
>
> GitHub には、その他のリソースもあります。完全な例を見つけて、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create CloudWatch service object
var cw = new AWS.CloudWatch({ apiVersion: "2010-08-01" });

var params = {
  AlarmName: "Web_Server_CPU_Utilization",
  ComparisonOperator: "GreaterThanThreshold",
  EvaluationPeriods: 1,
  MetricName: "CPUUtilization",
  Namespace: "AWS/EC2",
  Period: 60,
  Statistic: "Average",
  Threshold: 70.0,
```

```
  ActionsEnabled: false,
  AlarmDescription: "Alarm when server CPU exceeds 70%",
  Dimensions: [
    {
      Name: "InstanceId",
      Value: "INSTANCE_ID",
    },
  ],
  Unit: "Percent",
};

cw.putMetricAlarm(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- 詳細については、「AWS SDK for JavaScript デベロッパーガイド」を参照してください。
- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「PutMetricAlarm」を参照してください。

Kotlin

SDK for Kotlin

> **Note**
>
> GitHub には、その他のリソースもあります。完全な例を見つけて、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
suspend fun putMetricAlarm(alarmNameVal: String, instanceIdVal: String) {

    val dimensionOb = Dimension {
        name = "InstanceId"
        value = instanceIdVal
    }
```

```
    val request = PutMetricAlarmRequest {
        alarmName = alarmNameVal
        comparisonOperator = ComparisonOperator.GreaterThanThreshold
        evaluationPeriods = 1
        metricName = "CPUUtilization"
        namespace = "AWS/EC2"
        period = 60
        statistic = Statistic.fromValue("Average")
        threshold = 70.0
        actionsEnabled = false
        alarmDescription = "An Alarm created by the Kotlin SDK when server CPU
 utilization exceeds 70%"
        unit = StandardUnit.fromValue("Seconds")
        dimensions = listOf(dimensionOb)
    }

    CloudWatchClient { region = "us-east-1" }.use { cwClient ->
        cwClient.putMetricAlarm(request)
        println("Successfully created an alarm with name $alarmNameVal")
    }
}
```

- API の詳細については、AWS SDK for Kotlin API リファレンスの「PutMetricAlarm」を参照してください。

Python

SDK for Python (Boto3)

> **Note**
>
> GitHub には、その他のリソースもあります。完全な例を見つけて、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
class CloudWatchWrapper:
    """Encapsulates Amazon CloudWatch functions."""

    def __init__(self, cloudwatch_resource):
        """
```

```
        :param cloudwatch_resource: A Boto3 CloudWatch resource.
        """
        self.cloudwatch_resource = cloudwatch_resource


    def create_metric_alarm(
        self,
        metric_namespace,
        metric_name,
        alarm_name,
        stat_type,
        period,
        eval_periods,
        threshold,
        comparison_op,
    ):
        """
        Creates an alarm that watches a metric.

        :param metric_namespace: The namespace of the metric.
        :param metric_name: The name of the metric.
        :param alarm_name: The name of the alarm.
        :param stat_type: The type of statistic the alarm watches.
        :param period: The period in which metric data are grouped to calculate
                       statistics.
        :param eval_periods: The number of periods that the metric must be over
the
                             alarm threshold before the alarm is set into an
alarmed
                             state.
        :param threshold: The threshold value to compare against the metric
statistic.
        :param comparison_op: The comparison operation used to compare the
threshold
                              against the metric.
        :return: The newly created alarm.
        """
        try:
            metric = self.cloudwatch_resource.Metric(metric_namespace,
metric_name)
            alarm = metric.put_alarm(
                AlarmName=alarm_name,
                Statistic=stat_type,
                Period=period,
```

```
                EvaluationPeriods=eval_periods,
                Threshold=threshold,
                ComparisonOperator=comparison_op,
            )
            logger.info(
                "Added alarm %s to track metric %s.%s.",
                alarm_name,
                metric_namespace,
                metric_name,
            )
        except ClientError:
            logger.exception(
                "Couldn't add alarm %s to metric %s.%s",
                alarm_name,
                metric_namespace,
                metric_name,
            )
            raise
        else:
            return alarm
```

- API の詳細については、AWS SDK for Python (Boto3) API リファレンスの「PutMetricAlarm」を参照してください。

Ruby

SDK for Ruby

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
# Creates or updates an alarm in Amazon CloudWatch.
#
# @param cloudwatch_client [Aws::CloudWatch::Client]
#   An initialized CloudWatch client.
# @param alarm_name [String] The name of the alarm.
```

```
# @param alarm_description [String] A description about the alarm.
# @param metric_name [String] The name of the metric associated with the alarm.
# @param alarm_actions [Array] A list of Strings representing the
#   Amazon Resource Names (ARNs) to execute when the alarm transitions to the
#   ALARM state.
# @param namespace [String] The namespace for the metric to alarm on.
# @param statistic [String] The statistic for the metric.
# @param dimensions [Array] A list of dimensions for the metric, specified as
#   Aws::CloudWatch::Types::Dimension.
# @param period [Integer] The number of seconds before re-evaluating the metric.
# @param unit [String] The unit of measure for the statistic.
# @param evaluation_periods [Integer] The number of periods over which data is
#   compared to the specified threshold.
# @param theshold [Float] The value against which the specified statistic is
 compared.
# @param comparison_operator [String] The arithmetic operation to use when
#   comparing the specified statistic and threshold.
# @return [Boolean] true if the alarm was created or updated; otherwise, false.
# @example
#   exit 1 unless alarm_created_or_updated?(
#     Aws::CloudWatch::Client.new(region: 'us-east-1'),
#     'ObjectsInBucket',
#     'Objects exist in this bucket for more than 1 day.',
#     'NumberOfObjects',
#     ['arn:aws:sns:us-east-1:111111111111:Default_CloudWatch_Alarms_Topic'],
#     'AWS/S3',
#     'Average',
#     [
#       {
#         name: 'BucketName',
#         value: 'doc-example-bucket'
#       },
#       {
#         name: 'StorageType',
#         value: 'AllStorageTypes'
#       }
#     ],
#     86_400,
#     'Count',
#     1,
#     1,
#     'GreaterThanThreshold'
#   )
def alarm_created_or_updated?(
```

```
  cloudwatch_client,
  alarm_name,
  alarm_description,
  metric_name,
  alarm_actions,
  namespace,
  statistic,
  dimensions,
  period,
  unit,
  evaluation_periods,
  threshold,
  comparison_operator
)
  cloudwatch_client.put_metric_alarm(
    alarm_name: alarm_name,
    alarm_description: alarm_description,
    metric_name: metric_name,
    alarm_actions: alarm_actions,
    namespace: namespace,
    statistic: statistic,
    dimensions: dimensions,
    period: period,
    unit: unit,
    evaluation_periods: evaluation_periods,
    threshold: threshold,
    comparison_operator: comparison_operator
  )
  return true
rescue StandardError => e
  puts "Error creating alarm: #{e.message}"
  return false
end
```

- API の詳細については、「AWS SDK for Ruby API リファレンス」の「PutMetricAlarm」を参照してください。

SAP ABAP

SDK for SAP ABAP

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
TRY.
    lo_cwt->putmetricalarm(
      iv_alarmname                = iv_alarm_name
      iv_comparisonoperator       = iv_comparison_operator
      iv_evaluationperiods        = iv_evaluation_periods
      iv_metricname               = iv_metric_name
      iv_namespace                = iv_namespace
      iv_statistic                = iv_statistic
      iv_threshold                = iv_threshold
      iv_actionsenabled           = iv_actions_enabled
      iv_alarmdescription         = iv_alarm_description
      iv_unit                     = iv_unit
      iv_period                   = iv_period
      it_dimensions               = it_dimensions
    ).
    MESSAGE 'Alarm created.' TYPE 'I'.
  CATCH /aws1/cx_cwtlimitexceededfault.
    MESSAGE 'The request processing has exceeded the limit' TYPE 'E'.
ENDTRY.
```

- API の詳細については、「AWS SDK for SAP ABAP API リファレンス」の「PutMetricAlarm」を参照してください。

AWS SDK デベロッパーガイドとコード例の完全なリストについては、「AWS SDK で CloudWatch を使用する」を参照してください。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK または CLI で **PutMetricData** を使用する

以下のコード例は、PutMetricData の使用方法を示しています。

アクション例は、より大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。次のコード例で、このアクションのコンテキストを確認できます。

- メトリクス、ダッシュボード、およびアラームの使用を開始する
- メトリクスとアラームを管理する

.NET

AWS SDK for .NET

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Add some metric data using a call to a wrapper class.
    /// </summary>
    /// <param name="customMetricName">The metric name.</param>
    /// <param name="customMetricNamespace">The metric namespace.</param>
    /// <returns></returns>
    private static async Task<List<MetricDatum>> PutRandomMetricData(string
customMetricName,
        string customMetricNamespace)
    {
        List<MetricDatum> customData = new List<MetricDatum>();
        Random rnd = new Random();

        // Add 10 random values up to 100, starting with a timestamp 15 minutes
in the past.
        var utcNowMinus15 = DateTime.UtcNow.AddMinutes(-15);
        for (int i = 0; i < 10; i++)
        {
            var metricValue = rnd.Next(0, 100);
            customData.Add(
                new MetricDatum
                {
                    MetricName = customMetricName,
                    Value = metricValue,
```

```
                    TimestampUtc = utcNowMinus15.AddMinutes(i)
                }
            );
        }

        await _cloudWatchWrapper.PutMetricData(customMetricNamespace,
customData);
        return customData;
    }

    /// <summary>
    /// Wrapper to add metric data to a CloudWatch metric.
    /// </summary>
    /// <param name="metricNamespace">The namespace of the metric.</param>
    /// <param name="metricData">A data object for the metric data.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> PutMetricData(string metricNamespace,
        List<MetricDatum> metricData)
    {
        var putDataResponse = await _amazonCloudWatch.PutMetricDataAsync(
            new PutMetricDataRequest()
            {
                MetricData = metricData,
                Namespace = metricNamespace,
            });

        return putDataResponse.HttpStatusCode == HttpStatusCode.OK;
    }
```

- API の詳細については、「AWS SDK for .NET API リファレンス」の「PutMetricData」を参照してください。

C++

SDK for C++

> **Note**
>
> GitHub には、その他のリソースもあります。完全な例を見つけて、AWS コード例リポジトリでの設定と実行の方法を確認してください。

必要なファイルを含めます。

```
#include <aws/core/Aws.h>
#include <aws/monitoring/CloudWatchClient.h>
#include <aws/monitoring/model/PutMetricDataRequest.h>
#include <iostream>
```

メトリクスにデータを配置します。

```
        Aws::CloudWatch::CloudWatchClient cw;

        Aws::CloudWatch::Model::Dimension dimension;
        dimension.SetName("UNIQUE_PAGES");
        dimension.SetValue("URLS");

        Aws::CloudWatch::Model::MetricDatum datum;
        datum.SetMetricName("PAGES_VISITED");
        datum.SetUnit(Aws::CloudWatch::Model::StandardUnit::None);
        datum.SetValue(data_point);
        datum.AddDimensions(dimension);

        Aws::CloudWatch::Model::PutMetricDataRequest request;
        request.SetNamespace("SITE/TRAFFIC");
        request.AddMetricData(datum);

        auto outcome = cw.PutMetricData(request);
        if (!outcome.IsSuccess())
        {
            std::cout << "Failed to put sample metric data:" <<
                outcome.GetError().GetMessage() << std::endl;
        }
        else
        {
            std::cout << "Successfully put sample metric data" << std::endl;
        }
```

- API の詳細については、「AWS SDK for C++ API リファレンス」の「PutMetricData」を参照してください。

CLI

AWS CLI

Amazon CloudWatch にカスタムメトリクスをパブリッシュするには

次の例は、put-metric-data コマンドを使用して、Amazon CloudWatch にカスタムメトリクスをパブリッシュします。

```
aws cloudwatch put-metric-data --namespace "Usage Metrics" --metric-data file://
metric.json
```

メトリクス自体の値は、JSON ファイル metric.json に保存されます。

ファイルの内容は次のとおりです。

```
[
  {
    "MetricName": "New Posts",
    "Timestamp": "Wednesday, June 12, 2013 8:28:20 PM",
    "Value": 0.50,
    "Unit": "Count"
  }
]
```

詳細については、「Amazon CloudWatch ユーザーガイド」の「カスタムメトリクスをパブリッシュする」を参照してください。

複数のディメンションを指定するには

次の例は、複数のディメンションを指定する方法を示しています。各ディメンションは Name=Value ペアとして指定されます。複数のディメンションはコンマで区切ります。

```
aws cloudwatch put-metric-data --metric-name Buffers --namespace
 MyNameSpace --unit Bytes --value 231434333 --dimensions
 InstanceID=1-23456789,InstanceType=m1.small
```

- API の詳細については、AWS CLI コマンドリファレンスの「PutMetricData」を参照してください。

Java

SDK for Java 2.x

> **Note**
>
> GitHub には、その他のリソースもあります。完全な例を見つけて、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
   public static void addMetricDataForAlarm(CloudWatchClient cw, String
fileName) {
       try {
           // Read values from the JSON file.
           JsonParser parser = new JsonFactory().createParser(new
File(fileName));
           com.fasterxml.jackson.databind.JsonNode rootNode = new
ObjectMapper().readTree(parser);
           String customMetricNamespace =
rootNode.findValue("customMetricNamespace").asText();
           String customMetricName =
rootNode.findValue("customMetricName").asText();

           // Set an Instant object.
           String time =
ZonedDateTime.now(ZoneOffset.UTC).format(DateTimeFormatter.ISO_INSTANT);
           Instant instant = Instant.parse(time);

           MetricDatum datum = MetricDatum.builder()
                   .metricName(customMetricName)
                   .unit(StandardUnit.NONE)
                   .value(1001.00)
                   .timestamp(instant)
                   .build();

           MetricDatum datum2 = MetricDatum.builder()
                   .metricName(customMetricName)
                   .unit(StandardUnit.NONE)
                   .value(1002.00)
                   .timestamp(instant)
                   .build();
```

```
            List<MetricDatum> metricDataList = new ArrayList<>();
            metricDataList.add(datum);
            metricDataList.add(datum2);

            PutMetricDataRequest request = PutMetricDataRequest.builder()
                    .namespace(customMetricNamespace)
                    .metricData(metricDataList)
                    .build();

            cw.putMetricData(request);
            System.out.println("Added metric values for for metric " +
 customMetricName);

        } catch (CloudWatchException | IOException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }
```

- API の詳細については、「AWS SDK for Java 2.x API リファレンス」の「PutMetricData」を参照してください。

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

SDK モジュールとクライアントモジュールをインポートし、API を呼び出します。

```
import { PutMetricDataCommand } from "@aws-sdk/client-cloudwatch";
import { client } from "../libs/client.js";

const run = async () => {
  // See https://docs.aws.amazon.com/AmazonCloudWatch/latest/APIReference/
API_PutMetricData.html#API_PutMetricData_RequestParameters
```

```
  // and https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/
publishingMetrics.html
  // for more information about the parameters in this command.
  const command = new PutMetricDataCommand({
    MetricData: [
      {
        MetricName: "PAGES_VISITED",
        Dimensions: [
          {
            Name: "UNIQUE_PAGES",
            Value: "URLS",
          },
        ],
        Unit: "None",
        Value: 1.0,
      },
    ],
    Namespace: "SITE/TRAFFIC",
  });

  try {
    return await client.send(command);
  } catch (err) {
    console.error(err);
  }
};

export default run();
```

別のモジュールでクライアントを作成し、エクスポートします。

```
import { CloudWatchClient } from "@aws-sdk/client-cloudwatch";

export const client = new CloudWatchClient({});
```

- 詳細については、「AWS SDK for JavaScript デベロッパーガイド」を参照してください。
- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「PutMetricData」を参照してください。

SDK for JavaScript (v2)

> **Note**
>
> GitHub には、その他のリソースもあります。完全な例を見つけて、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create CloudWatch service object
var cw = new AWS.CloudWatch({ apiVersion: "2010-08-01" });

// Create parameters JSON for putMetricData
var params = {
  MetricData: [
    {
      MetricName: "PAGES_VISITED",
      Dimensions: [
        {
          Name: "UNIQUE_PAGES",
          Value: "URLS",
        },
      ],
      Unit: "None",
      Value: 1.0,
    },
  ],
  Namespace: "SITE/TRAFFIC",
};

cw.putMetricData(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", JSON.stringify(data));
  }
});
```

- 詳細については、「AWS SDK for JavaScript デベロッパーガイド」を参照してください。
- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「PutMetricData」を参照してください。

Kotlin

SDK for Kotlin

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
suspend fun addMetricDataForAlarm(fileName: String?) {
    // Read values from the JSON file.
    val parser = JsonFactory().createParser(File(fileName))
    val rootNode = ObjectMapper().readTree<JsonNode>(parser)
    val customMetricNamespace =
 rootNode.findValue("customMetricNamespace").asText()
    val customMetricName = rootNode.findValue("customMetricName").asText()

    // Set an Instant object.
    val time =
 ZonedDateTime.now(ZoneOffset.UTC).format(DateTimeFormatter.ISO_INSTANT)
    val instant = Instant.parse(time)
    val datum = MetricDatum {
        metricName = customMetricName
        unit = StandardUnit.None
        value = 1001.00
        timestamp = aws.smithy.kotlin.runtime.time.Instant(instant)
    }

    val datum2 = MetricDatum {
        metricName = customMetricName
        unit = StandardUnit.None
        value = 1002.00
        timestamp = aws.smithy.kotlin.runtime.time.Instant(instant)
    }

    val metricDataList = ArrayList<MetricDatum>()
```

```
    metricDataList.add(datum)
    metricDataList.add(datum2)

    val request = PutMetricDataRequest {
        namespace = customMetricNamespace
        metricData = metricDataList
    }

    CloudWatchClient { region = "us-east-1" }.use { cwClient ->
        cwClient.putMetricData(request)
        println("Added metric values for for metric $customMetricName")
    }
}
```

- API の詳細については、「AWS SDK for Kotlin API リファレンス」の「PutMetricData」を参照してください。

PowerShell

Tools for PowerShell

例 1: 新しい MetricDatum オブジェクトを作成し、Amazon Web Services の CloudWatch メトリクスに書き込みます。

```
### Create a MetricDatum .NET object
$Metric = New-Object -TypeName Amazon.CloudWatch.Model.MetricDatum
$Metric.Timestamp = [DateTime]::UtcNow
$Metric.MetricName = 'CPU'
$Metric.Value = 50

### Write the metric data to the CloudWatch service
Write-CWMetricData -Namespace instance1 -MetricData $Metric
```

- API の詳細については、「AWS Tools for PowerShell コマンドレットリファレンス」の「PutMetricData」を参照してください。

Python

SDK for Python (Boto3)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
class CloudWatchWrapper:
    """Encapsulates Amazon CloudWatch functions."""

    def __init__(self, cloudwatch_resource):
        """
        :param cloudwatch_resource: A Boto3 CloudWatch resource.
        """
        self.cloudwatch_resource = cloudwatch_resource


    def put_metric_data(self, namespace, name, value, unit):
        """
        Sends a single data value to CloudWatch for a metric. This metric is
 given
        a timestamp of the current UTC time.

        :param namespace: The namespace of the metric.
        :param name: The name of the metric.
        :param value: The value of the metric.
        :param unit: The unit of the metric.
        """
        try:
            metric = self.cloudwatch_resource.Metric(namespace, name)
            metric.put_data(
                Namespace=namespace,
                MetricData=[{"MetricName": name, "Value": value, "Unit": unit}],
            )
            logger.info("Put data for metric %s.%s", namespace, name)
        except ClientError:
            logger.exception("Couldn't put data for metric %s.%s", namespace,
 name)
            raise
```

一連のデータを CloudWatch メトリクスに配置します。

```
class CloudWatchWrapper:
    """Encapsulates Amazon CloudWatch functions."""

    def __init__(self, cloudwatch_resource):
        """
        :param cloudwatch_resource: A Boto3 CloudWatch resource.
        """
        self.cloudwatch_resource = cloudwatch_resource


    def put_metric_data_set(self, namespace, name, timestamp, unit, data_set):
        """
        Sends a set of data to CloudWatch for a metric. All of the data in the
 set
        have the same timestamp and unit.

        :param namespace: The namespace of the metric.
        :param name: The name of the metric.
        :param timestamp: The UTC timestamp for the metric.
        :param unit: The unit of the metric.
        :param data_set: The set of data to send. This set is a dictionary that
                         contains a list of values and a list of corresponding
 counts.
                         The value and count lists must be the same length.
        """
        try:
            metric = self.cloudwatch_resource.Metric(namespace, name)
            metric.put_data(
                Namespace=namespace,
                MetricData=[
                    {
                        "MetricName": name,
                        "Timestamp": timestamp,
                        "Values": data_set["values"],
                        "Counts": data_set["counts"],
                        "Unit": unit,
                    }
                ],
```

```
            )
            logger.info("Put data set for metric %s.%s.", namespace, name)
        except ClientError:
            logger.exception("Couldn't put data set for metric %s.%s.",
namespace, name)
            raise
```

- API の詳細については、AWS SDK for Python (Boto3) API リファレンスの「PutMetricData」を参照してください。

Ruby

SDK for Ruby

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
require "aws-sdk-cloudwatch"

# Adds a datapoint to a metric in Amazon CloudWatch.
#
# @param cloudwatch_client [Aws::CloudWatch::Client]
#   An initialized CloudWatch client.
# @param metric_namespace [String] The namespace of the metric to add the
#   datapoint to.
# @param metric_name [String] The name of the metric to add the datapoint to.
# @param dimension_name [String] The name of the dimension to add the
#   datapoint to.
# @param dimension_value [String] The value of the dimension to add the
#   datapoint to.
# @param metric_value [Float] The value of the datapoint.
# @param metric_unit [String] The unit of measurement for the datapoint.
# @return [Boolean]
# @example
#   exit 1 unless datapoint_added_to_metric?(
#     Aws::CloudWatch::Client.new(region: 'us-east-1'),
```

```
#     'SITE/TRAFFIC',
#     'UniqueVisitors',
#     'SiteName',
#     'example.com',
#     5_885.0,
#     'Count'
#   )
def datapoint_added_to_metric?(
  cloudwatch_client,
  metric_namespace,
  metric_name,
  dimension_name,
  dimension_value,
  metric_value,
  metric_unit
)
  cloudwatch_client.put_metric_data(
    namespace: metric_namespace,
    metric_data: [
      {
        metric_name: metric_name,
        dimensions: [
          {
            name: dimension_name,
            value: dimension_value
          }
        ],
        value: metric_value,
        unit: metric_unit
      }
    ]
  )
  puts "Added data about '#{metric_name}' to namespace " \
    "'#{metric_namespace}'."
  return true
rescue StandardError => e
  puts "Error adding data about '#{metric_name}' to namespace " \
    "'#{metric_namespace}': #{e.message}"
  return false
end
```

- API の詳細については、「AWS SDK for Ruby API リファレンス」の「PutMetricData」を参照してください。

AWS SDK デベロッパーガイドとコード例の完全なリストについては、「AWS SDK で CloudWatch を使用する」を参照してください。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK を使用した CloudWatch のシナリオ

以下のコード例は、一般的なシナリオを AWS SDK で CloudWatch に実装する方法を示します。これらのシナリオは、CloudWatch 内で複数の関数を呼び出して特定のタスクを実行する方法を示します。それぞれのシナリオには、GitHub へのリンクがあり、コードを設定および実行する方法についての説明です。

例

- AWS SDK を使用した CloudWatch アラームの開始方法
- AWS SDK を使用して CloudWatch のメトリクス、ダッシュボード、およびアラームの使用を開始する
- AWS SDK により CloudWatch メトリクスとアラームを管理する

AWS SDK を使用した CloudWatch アラームの開始方法

次のコードサンプルは、以下の操作方法を示しています。

- アラームを作成します。
- アラームアクションの無効化。
- アラームの記述。
- アラームの削除。

SAP ABAP

SDK for SAP ABAP

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
DATA lt_alarmnames TYPE /aws1/cl_cwtalarmnames_w=>tt_alarmnames.
DATA lo_alarmname TYPE REF TO /aws1/cl_cwtalarmnames_w.

"Create an alarm"
TRY.
    lo_cwt->putmetricalarm(
      iv_alarmname                = iv_alarm_name
      iv_comparisonoperator       = iv_comparison_operator
      iv_evaluationperiods        = iv_evaluation_periods
      iv_metricname               = iv_metric_name
      iv_namespace                = iv_namespace
      iv_statistic                = iv_statistic
      iv_threshold                = iv_threshold
      iv_actionsenabled           = iv_actions_enabled
      iv_alarmdescription         = iv_alarm_description
      iv_unit                     = iv_unit
      iv_period                   = iv_period
      it_dimensions               = it_dimensions
    ).
    MESSAGE 'Alarm created' TYPE 'I'.
  CATCH /aws1/cx_cwtlimitexceededfault.
    MESSAGE 'The request processing has exceeded the limit' TYPE 'E'.
ENDTRY.

"Create an ABAP internal table for the created alarm."
CREATE OBJECT lo_alarmname EXPORTING iv_value = iv_alarm_name.
INSERT lo_alarmname INTO TABLE lt_alarmnames.

"Disable alarm actions."
TRY.
    lo_cwt->disablealarmactions(
```

```
          it_alarmnames                = lt_alarmnames
        ).
        MESSAGE 'Alarm actions disabled' TYPE 'I'.
      CATCH /aws1/cx_rt_service_generic INTO DATA(lo_disablealarm_exception).
        DATA(lv_disablealarm_error) = |"{ lo_disablealarm_exception-
>av_err_code }" - { lo_disablealarm_exception->av_err_msg }|.
        MESSAGE lv_disablealarm_error TYPE 'E'.
    ENDTRY.

    "Describe alarm using the same ABAP internal table."
    TRY.
        oo_result = lo_cwt->describealarms(                      " oo_result is
 returned for testing purpose "
          it_alarmnames                = lt_alarmnames
        ).
        MESSAGE 'Alarms retrieved' TYPE 'I'.
      CATCH /aws1/cx_rt_service_generic INTO DATA(lo_describealarms_exception).
        DATA(lv_describealarms_error) = |"{ lo_describealarms_exception-
>av_err_code }" - { lo_describealarms_exception->av_err_msg }|.
        MESSAGE lv_describealarms_error TYPE 'E'.
    ENDTRY.

    "Delete alarm."
    TRY.
        lo_cwt->deletealarms(
          it_alarmnames = lt_alarmnames
        ).
        MESSAGE 'Alarms deleted' TYPE 'I'.
      CATCH /aws1/cx_cwtresourcenotfound .
        MESSAGE 'Resource being access is not found.' TYPE 'E'.
    ENDTRY.
```

- API の詳細については、「AWS SDK for SAP ABAP API リファレンス」の次のトピックを参照してください。
  - DeleteAlarms
  - DescribeAlarms
  - DisableAlarmActions
  - PutMetricAlarm

AWS SDK デベロッパーガイドとコード例の完全なリストについては、「AWS SDK で CloudWatch を使用する」を参照してください。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK を使用して CloudWatch のメトリクス、ダッシュボード、およびアラームの使用を開始する

次のコード例は、以下を実行する方法を示しています。

- CloudWatch の名前空間とメトリクスをリストする。
- メトリクスと予想請求額の統計の取得
- ダッシュボードの作成と更新
- メトリクスの作成とデータの追加
- アラームの作成/トリガーとアラーム履歴の表示
- 異常ディテクターの追加
- メトリクス画像の取得とリソースのクリーンアップ

.NET

AWS SDK for .NET

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

コマンドプロンプトからインタラクティブのシナリオを実行します。

```
public class CloudWatchScenario
{
    /*
    Before running this .NET code example, set up your development environment,
 including your credentials.

    To enable billing metrics and statistics for this example, make sure billing
 alerts are enabled for your account:
```

```
    https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/
monitor_estimated_charges_with_cloudwatch.html#turning_on_billing_metrics

    This .NET example performs the following tasks:
        1. List and select a CloudWatch namespace.
        2. List and select a CloudWatch metric.
        3. Get statistics for a CloudWatch metric.
        4. Get estimated billing statistics for the last week.
        5. Create a new CloudWatch dashboard with two metrics.
        6. List current CloudWatch dashboards.
        7. Create a CloudWatch custom metric and add metric data.
        8. Add the custom metric to the dashboard.
        9. Create a CloudWatch alarm for the custom metric.
       10. Describe current CloudWatch alarms.
       11. Get recent data for the custom metric.
       12. Add data to the custom metric to trigger the alarm.
       13. Wait for an alarm state.
       14. Get history for the CloudWatch alarm.
       15. Add an anomaly detector.
       16. Describe current anomaly detectors.
       17. Get and display a metric image.
       18. Clean up resources.
    */

    private static ILogger logger = null!;
    private static CloudWatchWrapper _cloudWatchWrapper = null!;
    private static IConfiguration _configuration = null!;
    private static readonly List<string> _statTypes = new List<string>
{ "SampleCount", "Average", "Sum", "Minimum", "Maximum" };
    private static SingleMetricAnomalyDetector? anomalyDetector = null!;

    static async Task Main(string[] args)
    {
        // Set up dependency injection for the Amazon service.
        using var host = Host.CreateDefaultBuilder(args)
            .ConfigureLogging(logging =>
                logging.AddFilter("System", LogLevel.Debug)
                    .AddFilter<DebugLoggerProvider>("Microsoft",
 LogLevel.Information)
                    .AddFilter<ConsoleLoggerProvider>("Microsoft",
 LogLevel.Trace))
            .ConfigureServices((_, services) =>
            services.AddAWSService<IAmazonCloudWatch>()
            .AddTransient<CloudWatchWrapper>()
```

```
        )
        .Build();

        _configuration = new ConfigurationBuilder()
            .SetBasePath(Directory.GetCurrentDirectory())
            .AddJsonFile("settings.json") // Load settings from .json file.
            .AddJsonFile("settings.local.json",
                true) // Optionally, load local settings.
            .Build();

        logger = LoggerFactory.Create(builder => { builder.AddConsole(); })
            .CreateLogger<CloudWatchScenario>();

        _cloudWatchWrapper =
host.Services.GetRequiredService<CloudWatchWrapper>();

        Console.WriteLine(new string('-', 80));
        Console.WriteLine("Welcome to the Amazon CloudWatch example scenario.");
        Console.WriteLine(new string('-', 80));

        try
        {
            var selectedNamespace = await SelectNamespace();
            var selectedMetric = await SelectMetric(selectedNamespace);
            await GetAndDisplayMetricStatistics(selectedNamespace,
selectedMetric);
            await GetAndDisplayEstimatedBilling();
            await CreateDashboardWithMetrics();
            await ListDashboards();
            await CreateNewCustomMetric();
            await AddMetricToDashboard();
            await CreateMetricAlarm();
            await DescribeAlarms();
            await GetCustomMetricData();
            await AddMetricDataForAlarm();
            await CheckForMetricAlarm();
            await GetAlarmHistory();
            anomalyDetector = await AddAnomalyDetector();
            await DescribeAnomalyDetectors();
            await GetAndOpenMetricImage();
            await CleanupResources();
        }
        catch (Exception ex)
        {
```

```
            logger.LogError(ex, "There was a problem executing the scenario.");
            await CleanupResources();
        }

    }

    /// <summary>
    /// Select a namespace.
    /// </summary>
    /// <returns>The selected namespace.</returns>
    private static async Task<string> SelectNamespace()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"1. Select a CloudWatch Namespace from a list of
Namespaces.");
        var metrics = await _cloudWatchWrapper.ListMetrics();
        // Get a distinct list of namespaces.
        var namespaces = metrics.Select(m => m.Namespace).Distinct().ToList();
        for (int i = 0; i < namespaces.Count; i++)
        {
            Console.WriteLine($"\t{i + 1}. {namespaces[i]}");
        }

        var namespaceChoiceNumber = 0;
        while (namespaceChoiceNumber < 1 || namespaceChoiceNumber >
namespaces.Count)
        {
            Console.WriteLine(
                "Select a namespace by entering a number from the preceding
list:");
            var choice = Console.ReadLine();
            Int32.TryParse(choice, out namespaceChoiceNumber);
        }

        var selectedNamespace = namespaces[namespaceChoiceNumber - 1];

        Console.WriteLine(new string('-', 80));

        return selectedNamespace;
    }

    /// <summary>
    /// Select a metric from a namespace.
    /// </summary>
```

```
    /// <param name="metricNamespace">The namespace for metrics.</param>
    /// <returns>The metric name.</returns>
    private static async Task<Metric> SelectMetric(string metricNamespace)
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"2. Select a CloudWatch metric from a namespace.");

        var namespaceMetrics = await
_cloudWatchWrapper.ListMetrics(metricNamespace);

        for (int i = 0; i < namespaceMetrics.Count && i < 15; i++)
        {
            var dimensionsWithValues = namespaceMetrics[i].Dimensions
                .Where(d => !string.Equals("None", d.Value));
            Console.WriteLine($"\t{i + 1}. {namespaceMetrics[i].MetricName} " +
                              $"{string.Join(", :", dimensionsWithValues.Select(d
=> d.Value))}");
        }

        var metricChoiceNumber = 0;
        while (metricChoiceNumber < 1 || metricChoiceNumber >
namespaceMetrics.Count)
        {
            Console.WriteLine(
                "Select a metric by entering a number from the preceding list:");
            var choice = Console.ReadLine();
            Int32.TryParse(choice, out metricChoiceNumber);
        }

        var selectedMetric = namespaceMetrics[metricChoiceNumber - 1];

        Console.WriteLine(new string('-', 80));

        return selectedMetric;
    }

    /// <summary>
    /// Get and display metric statistics for a specific metric.
    /// </summary>
    /// <param name="metricNamespace">The namespace for metrics.</param>
    /// <param name="metric">The CloudWatch metric.</param>
    /// <returns>Async task.</returns>
    private static async Task GetAndDisplayMetricStatistics(string
metricNamespace, Metric metric)
```

```
   {
       Console.WriteLine(new string('-', 80));
       Console.WriteLine($"3. Get CloudWatch metric statistics for the last
day.");

       for (int i = 0; i < _statTypes.Count; i++)
       {
           Console.WriteLine($"\t{i + 1}. {_statTypes[i]}");
       }

       var statisticChoiceNumber = 0;
       while (statisticChoiceNumber < 1 || statisticChoiceNumber >
_statTypes.Count)
       {
           Console.WriteLine(
               "Select a metric statistic by entering a number from the
preceding list:");
           var choice = Console.ReadLine();
           Int32.TryParse(choice, out statisticChoiceNumber);
       }

       var selectedStatistic = _statTypes[statisticChoiceNumber - 1];
       var statisticsList = new List<string> { selectedStatistic };

       var metricStatistics = await
_cloudWatchWrapper.GetMetricStatistics(metricNamespace, metric.MetricName,
statisticsList, metric.Dimensions, 1, 60);

       if (!metricStatistics.Any())
       {
           Console.WriteLine($"No {selectedStatistic} statistics found for
{metric} in namespace {metricNamespace}.");
       }

       metricStatistics = metricStatistics.OrderBy(s => s.Timestamp).ToList();
       for (int i = 0; i < metricStatistics.Count && i < 10; i++)
       {
           var metricStat = metricStatistics[i];
           var statValue =
metricStat.GetType().GetProperty(selectedStatistic)!.GetValue(metricStat, null);
           Console.WriteLine($"\t{i + 1}. Timestamp
{metricStatistics[i].Timestamp:G} {selectedStatistic}: {statValue}");
       }
```

```
        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Get and display estimated billing statistics.
    /// </summary>
    /// <param name="metricNamespace">The namespace for metrics.</param>
    /// <param name="metric">The CloudWatch metric.</param>
    /// <returns>Async task.</returns>
    private static async Task GetAndDisplayEstimatedBilling()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"4. Get CloudWatch estimated billing for the last
week.");

        var billingStatistics = await SetupBillingStatistics();

        for (int i = 0; i < billingStatistics.Count; i++)
        {
            Console.WriteLine($"\t{i + 1}. Timestamp
{billingStatistics[i].Timestamp:G} : {billingStatistics[i].Maximum}");
        }

        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Get billing statistics using a call to a wrapper class.
    /// </summary>
    /// <returns>A collection of billing statistics.</returns>
    private static async Task<List<Datapoint>> SetupBillingStatistics()
    {
        // Make a request for EstimatedCharges with a period of one day for the
past seven days.
        var billingStatistics = await _cloudWatchWrapper.GetMetricStatistics(
            "AWS/Billing",
            "EstimatedCharges",
            new List<string>() { "Maximum" },
            new List<Dimension>() { new Dimension { Name = "Currency", Value =
"USD" } },
            7,
            86400);

        billingStatistics = billingStatistics.OrderBy(n => n.Timestamp).ToList();
```

```
        return billingStatistics;
    }

    /// <summary>
    /// Create a dashboard with metrics.
    /// </summary>
    /// <param name="metricNamespace">The namespace for metrics.</param>
    /// <param name="metric">The CloudWatch metric.</param>
    /// <returns>Async task.</returns>
    private static async Task CreateDashboardWithMetrics()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"5. Create a new CloudWatch dashboard with metrics.");
        var dashboardName = _configuration["dashboardName"];
        var newDashboard = new DashboardModel();
        _configuration.GetSection("dashboardExampleBody").Bind(newDashboard);
        var newDashboardString = JsonSerializer.Serialize(
            newDashboard,
            new JsonSerializerOptions
            {
                DefaultIgnoreCondition = JsonIgnoreCondition.WhenWritingNull
            });
        var validationMessages =
            await _cloudWatchWrapper.PutDashboard(dashboardName,
newDashboardString);

        Console.WriteLine(validationMessages.Any() ? $"\tValidation messages:" :
null);
        for (int i = 0; i < validationMessages.Count; i++)
        {
            Console.WriteLine($"\t{i + 1}. {validationMessages[i].Message}");
        }
        Console.WriteLine($"\tDashboard {dashboardName} was created.");
        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// List dashboards.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task ListDashboards()
    {
        Console.WriteLine(new string('-', 80));
```

```
        Console.WriteLine($"6. List the CloudWatch dashboards in the current
account.");

        var dashboards = await _cloudWatchWrapper.ListDashboards();

        for (int i = 0; i < dashboards.Count; i++)
        {
            Console.WriteLine($"\t{i + 1}. {dashboards[i].DashboardName}");
        }

        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Create and add data for a new custom metric.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task CreateNewCustomMetric()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"7. Create and add data for a new custom metric.");

        var customMetricNamespace = _configuration["customMetricNamespace"];
        var customMetricName = _configuration["customMetricName"];

        var customData = await PutRandomMetricData(customMetricName,
customMetricNamespace);

        var valuesString = string.Join(',', customData.Select(d => d.Value));
        Console.WriteLine($"\tAdded metric values for for metric
{customMetricName}: \n\t{valuesString}");

        Console.WriteLine(new string('-', 80));
    }


    /// <summary>
    /// Add some metric data using a call to a wrapper class.
    /// </summary>
    /// <param name="customMetricName">The metric name.</param>
    /// <param name="customMetricNamespace">The metric namespace.</param>
    /// <returns></returns>
    private static async Task<List<MetricDatum>> PutRandomMetricData(string
customMetricName,
```

```
        string customMetricNamespace)
    {
        List<MetricDatum> customData = new List<MetricDatum>();
        Random rnd = new Random();

        // Add 10 random values up to 100, starting with a timestamp 15 minutes
in the past.
        var utcNowMinus15 = DateTime.UtcNow.AddMinutes(-15);
        for (int i = 0; i < 10; i++)
        {
            var metricValue = rnd.Next(0, 100);
            customData.Add(
                new MetricDatum
                {
                    MetricName = customMetricName,
                    Value = metricValue,
                    TimestampUtc = utcNowMinus15.AddMinutes(i)
                }
            );
        }

        await _cloudWatchWrapper.PutMetricData(customMetricNamespace,
customData);
        return customData;
    }

    /// <summary>
    /// Add the custom metric to the dashboard.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task AddMetricToDashboard()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"8. Add the new custom metric to the dashboard.");

        var dashboardName = _configuration["dashboardName"];

        var customMetricNamespace = _configuration["customMetricNamespace"];
        var customMetricName = _configuration["customMetricName"];

        var validationMessages = await SetupDashboard(customMetricNamespace,
customMetricName, dashboardName);
```

```
        Console.WriteLine(validationMessages.Any() ? $"\tValidation messages:" :
null);
        for (int i = 0; i < validationMessages.Count; i++)
        {
            Console.WriteLine($"\t{i + 1}. {validationMessages[i].Message}");
        }
        Console.WriteLine($"\tDashboard {dashboardName} updated with metric
{customMetricName}.");
        Console.WriteLine(new string('-', 80));
    }


    /// <summary>
    /// Set up a dashboard using a call to the wrapper class.
    /// </summary>
    /// <param name="customMetricNamespace">The metric namespace.</param>
    /// <param name="customMetricName">The metric name.</param>
    /// <param name="dashboardName">The name of the dashboard.</param>
    /// <returns>A list of validation messages.</returns>
    private static async Task<List<DashboardValidationMessage>> SetupDashboard(
        string customMetricNamespace, string customMetricName, string
dashboardName)
    {
        // Get the dashboard model from configuration.
        var newDashboard = new DashboardModel();
        _configuration.GetSection("dashboardExampleBody").Bind(newDashboard);

        // Add a new metric to the dashboard.
        newDashboard.Widgets.Add(new Widget
        {
            Height = 8,
            Width = 8,
            Y = 8,
            X = 0,
            Type = "metric",
            Properties = new Properties
            {
                Metrics = new List<List<object>>
                    { new() { customMetricNamespace, customMetricName } },
                View = "timeSeries",
                Region = "us-east-1",
                Stat = "Sum",
                Period = 86400,
                YAxis = new YAxis { Left = new Left { Min = 0, Max = 100 } },
```

```
                Title = "Custom Metric Widget",
                LiveData = true,
                Sparkline = true,
                Trend = true,
                Stacked = false,
                SetPeriodToTimeRange = false
            }
        });

        var newDashboardString = JsonSerializer.Serialize(newDashboard,
            new JsonSerializerOptions
            { DefaultIgnoreCondition = JsonIgnoreCondition.WhenWritingNull });
        var validationMessages =
            await _cloudWatchWrapper.PutDashboard(dashboardName,
newDashboardString);

        return validationMessages;
    }

    /// <summary>
    /// Create a CloudWatch alarm for the new metric.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task CreateMetricAlarm()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"9. Create a CloudWatch alarm for the new metric.");

        var customMetricNamespace = _configuration["customMetricNamespace"];
        var customMetricName = _configuration["customMetricName"];

        var alarmName = _configuration["exampleAlarmName"];
        var accountId = _configuration["accountId"];
        var region = _configuration["region"];
        var emailTopic = _configuration["emailTopic"];
        var alarmActions = new List<string>();

        if (GetYesNoResponse(
                $"\tAdd an email action for topic {emailTopic} to alarm
{alarmName}? (y/n)"))
        {
            _cloudWatchWrapper.AddEmailAlarmAction(accountId, region, emailTopic,
alarmActions);
        }
```

```
        await _cloudWatchWrapper.PutMetricEmailAlarm(
            "Example metric alarm",
            alarmName,
            ComparisonOperator.GreaterThanOrEqualToThreshold,
            customMetricName,
            customMetricNamespace,
            100,
            alarmActions);

        Console.WriteLine($"\tAlarm {alarmName} added for metric
{customMetricName}.");
        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Describe Alarms.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task DescribeAlarms()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"10. Describe CloudWatch alarms in the current
account.");

        var alarms = await _cloudWatchWrapper.DescribeAlarms();
        alarms = alarms.OrderByDescending(a => a.StateUpdatedTimestamp).ToList();

        for (int i = 0; i < alarms.Count && i < 10; i++)
        {
            var alarm = alarms[i];
            Console.WriteLine($"\t{i + 1}. {alarm.AlarmName}");
            Console.WriteLine($"\tState: {alarm.StateValue} for
{alarm.MetricName} {alarm.ComparisonOperator} {alarm.Threshold}");
        }

        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Get the recent data for the metric.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task GetCustomMetricData()
```

```
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"11. Get current data for new custom metric.");

        var customMetricNamespace = _configuration["customMetricNamespace"];
        var customMetricName = _configuration["customMetricName"];
        var accountId = _configuration["accountId"];

        var query = new List<MetricDataQuery>
        {
            new MetricDataQuery
            {
                AccountId = accountId,
                Id = "m1",
                Label = "Custom Metric Data",
                MetricStat = new MetricStat
                {
                    Metric = new Metric
                    {
                        MetricName = customMetricName,
                        Namespace = customMetricNamespace,
                    },
                    Period = 1,
                    Stat = "Maximum"
                }
            }
        };

        var metricData = await _cloudWatchWrapper.GetMetricData(
            20,
            true,
            DateTime.UtcNow.AddMinutes(1),
            20,
            query);

        for (int i = 0; i < metricData.Count; i++)
        {
            for (int j = 0; j < metricData[i].Values.Count; j++)
            {
                Console.WriteLine(
                    $"\tTimestamp {metricData[i].Timestamps[j]:G} Value:
{metricData[i].Values[j]}");
            }
        }
```

```
        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Add metric data to trigger an alarm.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task AddMetricDataForAlarm()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"12. Add metric data to the custom metric to trigger
an alarm.");

        var customMetricNamespace = _configuration["customMetricNamespace"];
        var customMetricName = _configuration["customMetricName"];
        var nowUtc = DateTime.UtcNow;
        List<MetricDatum> customData = new List<MetricDatum>
        {
            new MetricDatum
            {
                MetricName = customMetricName,
                Value = 101,
                TimestampUtc = nowUtc.AddMinutes(-2)
            },
            new MetricDatum
            {
                MetricName = customMetricName,
                Value = 101,
                TimestampUtc = nowUtc.AddMinutes(-1)
            },
            new MetricDatum
            {
                MetricName = customMetricName,
                Value = 101,
                TimestampUtc = nowUtc
            }
        };
        var valuesString = string.Join(',', customData.Select(d => d.Value));
        Console.WriteLine($"\tAdded metric values for for metric
{customMetricName}: \n\t{valuesString}");
        await _cloudWatchWrapper.PutMetricData(customMetricNamespace,
customData);
```

```
        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Check for a metric alarm using the DescribeAlarmsForMetric action.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task CheckForMetricAlarm()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"13. Checking for an alarm state.");

        var customMetricNamespace = _configuration["customMetricNamespace"];
        var customMetricName = _configuration["customMetricName"];
        var hasAlarm = false;
        var retries = 10;
        while (!hasAlarm && retries > 0)
        {
            var alarms = await
_cloudWatchWrapper.DescribeAlarmsForMetric(customMetricNamespace,
customMetricName);
            hasAlarm = alarms.Any(a => a.StateValue == StateValue.ALARM);
            retries--;
            Thread.Sleep(20000);
        }

        Console.WriteLine(hasAlarm
            ? $"\tAlarm state found for {customMetricName}."
            : $"\tNo Alarm state found for {customMetricName} after 10
retries.");

        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Get history for an alarm.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task GetAlarmHistory()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"14. Get alarm history.");

        var exampleAlarmName = _configuration["exampleAlarmName"];
```

```
        var alarmHistory = await
_cloudWatchWrapper.DescribeAlarmHistory(exampleAlarmName, 2);

        for (int i = 0; i < alarmHistory.Count; i++)
        {
            var history = alarmHistory[i];
            Console.WriteLine($"\t{i + 1}. {history.HistorySummary}, time
{history.Timestamp:g}");
        }
        if (!alarmHistory.Any())
        {
            Console.WriteLine($"\tNo alarm history data found for
{exampleAlarmName}.");
        }

        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Add an anomaly detector.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task<SingleMetricAnomalyDetector> AddAnomalyDetector()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"15. Add an anomaly detector.");

        var customMetricNamespace = _configuration["customMetricNamespace"];
        var customMetricName = _configuration["customMetricName"];

        var detector = new SingleMetricAnomalyDetector
        {
            MetricName = customMetricName,
            Namespace = customMetricNamespace,
            Stat = "Maximum"
        };
        await _cloudWatchWrapper.PutAnomalyDetector(detector);
        Console.WriteLine($"\tAdded anomaly detector for metric
{customMetricName}.");

        Console.WriteLine(new string('-', 80));
        return detector;
    }
```

```
    /// <summary>
    /// Describe anomaly detectors.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task DescribeAnomalyDetectors()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"16. Describe anomaly detectors in the current
account.");

        var customMetricNamespace = _configuration["customMetricNamespace"];
        var customMetricName = _configuration["customMetricName"];

        var detectors = await
_cloudWatchWrapper.DescribeAnomalyDetectors(customMetricNamespace,
customMetricName);

        for (int i = 0; i < detectors.Count; i++)
        {
            var detector = detectors[i];
            Console.WriteLine($"\t{i + 1}.
{detector.SingleMetricAnomalyDetector.MetricName}, state
{detector.StateValue}");
        }

        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Fetch and open a metrics image for a CloudWatch metric and namespace.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task GetAndOpenMetricImage()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine("17. Get a metric image from CloudWatch.");

        Console.WriteLine($"\tGetting Image data for custom metric.");
        var customMetricNamespace = _configuration["customMetricNamespace"];
        var customMetricName = _configuration["customMetricName"];
```

```
        var memoryStream = await
_cloudWatchWrapper.GetTimeSeriesMetricImage(customMetricNamespace,
customMetricName, "Maximum", 10);
        var file = _cloudWatchWrapper.SaveMetricImage(memoryStream,
"MetricImages");

        ProcessStartInfo info = new ProcessStartInfo();

        Console.WriteLine($"\tFile saved as {Path.GetFileName(file)}.");
        Console.WriteLine($"\tPress enter to open the image.");
        Console.ReadLine();
        info.FileName = Path.Combine("ms-photos://", file);
        info.UseShellExecute = true;
        info.CreateNoWindow = true;
        info.Verb = string.Empty;

        Process.Start(info);

        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Clean up created resources.
    /// </summary>
    /// <param name="metricNamespace">The namespace for metrics.</param>
    /// <param name="metric">The CloudWatch metric.</param>
    /// <returns>Async task.</returns>
    private static async Task CleanupResources()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"18. Clean up resources.");

        var dashboardName = _configuration["dashboardName"];
        if (GetYesNoResponse($"\tDelete dashboard {dashboardName}? (y/n)"))
        {
            Console.WriteLine($"\tDeleting dashboard.");
            var dashboardList = new List<string> { dashboardName };
            await _cloudWatchWrapper.DeleteDashboards(dashboardList);
        }

        var alarmName = _configuration["exampleAlarmName"];
        if (GetYesNoResponse($"\tDelete alarm {alarmName}? (y/n)"))
        {
            Console.WriteLine($"\tCleaning up alarms.");
```

```
            var alarms = new List<string> { alarmName };
            await _cloudWatchWrapper.DeleteAlarms(alarms);
        }

        if (GetYesNoResponse($"\tDelete anomaly detector? (y/n)") &&
 anomalyDetector != null)
        {
            Console.WriteLine($"\tCleaning up anomaly detector.");

            await _cloudWatchWrapper.DeleteAnomalyDetector(
                anomalyDetector);
        }

        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Get a yes or no response from the user.
    /// </summary>
    /// <param name="question">The question string to print on the console.</
param>
    /// <returns>True if the user responds with a yes.</returns>
    private static bool GetYesNoResponse(string question)
    {
        Console.WriteLine(question);
        var ynResponse = Console.ReadLine();
        var response = ynResponse != null &&
                       ynResponse.Equals("y",
                           StringComparison.InvariantCultureIgnoreCase);
        return response;
    }
}
```

CloudWatch アクションのシナリオで使用されるラッパーメソッドです。

```
/// <summary>
/// Wrapper class for Amazon CloudWatch methods.
/// </summary>
public class CloudWatchWrapper
{
    private readonly IAmazonCloudWatch _amazonCloudWatch;
    private readonly ILogger<CloudWatchWrapper> _logger;
```

```
    /// <summary>
    /// Constructor for the CloudWatch wrapper.
    /// </summary>
    /// <param name="amazonCloudWatch">The injected CloudWatch client.</param>
    /// <param name="logger">The injected logger for the wrapper.</param>
    public CloudWatchWrapper(IAmazonCloudWatch amazonCloudWatch,
ILogger<CloudWatchWrapper> logger)

    {
        _logger = logger;
        _amazonCloudWatch = amazonCloudWatch;
    }

    /// <summary>
    /// List metrics available, optionally within a namespace.
    /// </summary>
    /// <param name="metricNamespace">Optional CloudWatch namespace to use when
listing metrics.</param>
    /// <param name="filter">Optional dimension filter.</param>
    /// <param name="metricName">Optional metric name filter.</param>
    /// <returns>The list of metrics.</returns>
    public async Task<List<Metric>> ListMetrics(string? metricNamespace = null,
DimensionFilter? filter = null, string? metricName = null)
    {
        var results = new List<Metric>();
        var paginateMetrics = _amazonCloudWatch.Paginators.ListMetrics(
            new ListMetricsRequest
            {
                Namespace = metricNamespace,
                Dimensions = filter != null ? new List<DimensionFilter>
{ filter } : null,
                MetricName = metricName
            });
        // Get the entire list using the paginator.
        await foreach (var metric in paginateMetrics.Metrics)
        {
            results.Add(metric);
        }

        return results;
    }

    /// <summary>
```

```
    /// Wrapper to get statistics for a specific CloudWatch metric.
    /// </summary>
    /// <param name="metricNamespace">The namespace of the metric.</param>
    /// <param name="metricName">The name of the metric.</param>
    /// <param name="statistics">The list of statistics to include.</param>
    /// <param name="dimensions">The list of dimensions to include.</param>
    /// <param name="days">The number of days in the past to include.</param>
    /// <param name="period">The period for the data.</param>
    /// <returns>A list of DataPoint objects for the statistics.</returns>
    public async Task<List<Datapoint>> GetMetricStatistics(string
 metricNamespace,
        string metricName, List<string> statistics, List<Dimension> dimensions,
 int days, int period)
    {
        var metricStatistics = await _amazonCloudWatch.GetMetricStatisticsAsync(
            new GetMetricStatisticsRequest()
            {
                Namespace = metricNamespace,
                MetricName = metricName,
                Dimensions = dimensions,
                Statistics = statistics,
                StartTimeUtc = DateTime.UtcNow.AddDays(-days),
                EndTimeUtc = DateTime.UtcNow,
                Period = period
            });

        return metricStatistics.Datapoints;
    }

    /// <summary>
    /// Wrapper to create or add to a dashboard with metrics.
    /// </summary>
    /// <param name="dashboardName">The name for the dashboard.</param>
    /// <param name="dashboardBody">The metric data in JSON for the dashboard.</
param>
    /// <returns>A list of validation messages for the dashboard.</returns>
    public async Task<List<DashboardValidationMessage>> PutDashboard(string
 dashboardName,
        string dashboardBody)
    {
        // Updating a dashboard replaces all contents.
        // Best practice is to include a text widget indicating this dashboard
 was created programmatically.
        var dashboardResponse = await _amazonCloudWatch.PutDashboardAsync(
```

```
            new PutDashboardRequest()
            {
                DashboardName = dashboardName,
                DashboardBody = dashboardBody
            });

        return dashboardResponse.DashboardValidationMessages;
    }


    /// <summary>
    /// Get information on a dashboard.
    /// </summary>
    /// <param name="dashboardName">The name of the dashboard.</param>
    /// <returns>A JSON object with dashboard information.</returns>
    public async Task<string> GetDashboard(string dashboardName)
    {
        var dashboardResponse = await _amazonCloudWatch.GetDashboardAsync(
            new GetDashboardRequest()
            {
                DashboardName = dashboardName
            });

        return dashboardResponse.DashboardBody;
    }


    /// <summary>
    /// Get a list of dashboards.
    /// </summary>
    /// <returns>A list of DashboardEntry objects.</returns>
    public async Task<List<DashboardEntry>> ListDashboards()
    {
        var results = new List<DashboardEntry>();
        var paginateDashboards = _amazonCloudWatch.Paginators.ListDashboards(
            new ListDashboardsRequest());
        // Get the entire list using the paginator.
        await foreach (var data in paginateDashboards.DashboardEntries)
        {
            results.Add(data);
        }

        return results;
    }
```

```
    /// <summary>
    /// Wrapper to add metric data to a CloudWatch metric.
    /// </summary>
    /// <param name="metricNamespace">The namespace of the metric.</param>
    /// <param name="metricData">A data object for the metric data.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> PutMetricData(string metricNamespace,
        List<MetricDatum> metricData)
    {
        var putDataResponse = await _amazonCloudWatch.PutMetricDataAsync(
            new PutMetricDataRequest()
            {
                MetricData = metricData,
                Namespace = metricNamespace,
            });

        return putDataResponse.HttpStatusCode == HttpStatusCode.OK;
    }

    /// <summary>
    /// Get an image for a metric graphed over time.
    /// </summary>
    /// <param name="metricNamespace">The namespace of the metric.</param>
    /// <param name="metric">The name of the metric.</param>
    /// <param name="stat">The name of the stat to chart.</param>
    /// <param name="period">The period to use for the chart.</param>
    /// <returns>A memory stream for the chart image.</returns>
    public async Task<MemoryStream> GetTimeSeriesMetricImage(string
metricNamespace, string metric, string stat, int period)
    {
        var metricImageWidget = new
        {
            title = "Example Metric Graph",
            view = "timeSeries",
            stacked = false,
            period = period,
            width = 1400,
            height = 600,
            metrics = new List<List<object>>
                { new() { metricNamespace, metric, new { stat } } }
        };
```

```
        var metricImageWidgetString =
 JsonSerializer.Serialize(metricImageWidget);
        var imageResponse = await _amazonCloudWatch.GetMetricWidgetImageAsync(
            new GetMetricWidgetImageRequest()
            {
                MetricWidget = metricImageWidgetString
            });

        return imageResponse.MetricWidgetImage;
    }

    /// <summary>
    /// Save a metric image to a file.
    /// </summary>
    /// <param name="memoryStream">The MemoryStream for the metric image.</param>
    /// <param name="metricName">The name of the metric.</param>
    /// <returns>The path to the file.</returns>
    public string SaveMetricImage(MemoryStream memoryStream, string metricName)
    {
        var metricFileName = $"{metricName}_{DateTime.Now.Ticks}.png";
        using var sr = new StreamReader(memoryStream);
        // Writes the memory stream to a file.
        File.WriteAllBytes(metricFileName, memoryStream.ToArray());
        var filePath = Path.Join(AppDomain.CurrentDomain.BaseDirectory,
            metricFileName);
        return filePath;
    }

    /// <summary>
    /// Get data for CloudWatch metrics.
    /// </summary>
    /// <param name="minutesOfData">The number of minutes of data to include.</
param>
    /// <param name="useDescendingTime">True to return the data descending by
 time.</param>
    /// <param name="endDateUtc">The end date for the data, in UTC.</param>
    /// <param name="maxDataPoints">The maximum data points to include.</param>
    /// <param name="dataQueries">Optional data queries to include.</param>
    /// <returns>A list of the requested metric data.</returns>
    public async Task<List<MetricDataResult>> GetMetricData(int minutesOfData,
 bool useDescendingTime, DateTime? endDateUtc = null,
        int maxDataPoints = 0, List<MetricDataQuery>? dataQueries = null)
    {
        var metricData = new List<MetricDataResult>();
```

```
        // If no end time is provided, use the current time for the end time.
        endDateUtc ??= DateTime.UtcNow;
        var timeZoneOffset =
 TimeZoneInfo.Local.GetUtcOffset(endDateUtc.Value.ToLocalTime());
        var startTimeUtc = endDateUtc.Value.AddMinutes(-minutesOfData);
        // The timezone string should be in the format +0000, so use the timezone
 offset to format it correctly.
        var timeZoneString = $"{timeZoneOffset.Hours:D2}
{timeZoneOffset.Minutes:D2}";
        var paginatedMetricData = _amazonCloudWatch.Paginators.GetMetricData(
            new GetMetricDataRequest()
            {
                StartTimeUtc = startTimeUtc,
                EndTimeUtc = endDateUtc.Value,
                LabelOptions = new LabelOptions { Timezone = timeZoneString },
                ScanBy = useDescendingTime ? ScanBy.TimestampDescending :
 ScanBy.TimestampAscending,
                MaxDatapoints = maxDataPoints,
                MetricDataQueries = dataQueries,
            });

        await foreach (var data in paginatedMetricData.MetricDataResults)
        {
            metricData.Add(data);
        }
        return metricData;
    }

    /// <summary>
    /// Add a metric alarm to send an email when the metric passes a threshold.
    /// </summary>
    /// <param name="alarmDescription">A description of the alarm.</param>
    /// <param name="alarmName">The name for the alarm.</param>
    /// <param name="comparison">The type of comparison to use.</param>
    /// <param name="metricName">The name of the metric for the alarm.</param>
    /// <param name="metricNamespace">The namespace of the metric.</param>
    /// <param name="threshold">The threshold value for the alarm.</param>
    /// <param name="alarmActions">Optional actions to execute when in an alarm
 state.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> PutMetricEmailAlarm(string alarmDescription, string
 alarmName, ComparisonOperator comparison,
        string metricName, string metricNamespace, double threshold, List<string>
 alarmActions = null!)
```

```
    {
        try
        {
            var putEmailAlarmResponse = await
_amazonCloudWatch.PutMetricAlarmAsync(
                new PutMetricAlarmRequest()
                {
                    AlarmActions = alarmActions,
                    AlarmDescription = alarmDescription,
                    AlarmName = alarmName,
                    ComparisonOperator = comparison,
                    Threshold = threshold,
                    Namespace = metricNamespace,
                    MetricName = metricName,
                    EvaluationPeriods = 1,
                    Period = 10,
                    Statistic = new Statistic("Maximum"),
                    DatapointsToAlarm = 1,
                    TreatMissingData = "ignore"
                });
            return putEmailAlarmResponse.HttpStatusCode == HttpStatusCode.OK;
        }
        catch (LimitExceededException lex)
        {
            _logger.LogError(lex, $"Unable to add alarm {alarmName}. Alarm quota
has already been reached.");
        }

        return false;
    }

    /// <summary>
    /// Add specific email actions to a list of action strings for a CloudWatch
alarm.
    /// </summary>
    /// <param name="accountId">The AccountId for the alarm.</param>
    /// <param name="region">The region for the alarm.</param>
    /// <param name="emailTopicName">An Amazon Simple Notification Service (SNS)
topic for the alarm email.</param>
    /// <param name="alarmActions">Optional list of existing alarm actions to
append to.</param>
    /// <returns>A list of string actions for an alarm.</returns>
    public List<string> AddEmailAlarmAction(string accountId, string region,
        string emailTopicName, List<string>? alarmActions = null)
```

```
    {
        alarmActions ??= new List<string>();
        var snsAlarmAction = $"arn:aws:sns:{region}:{accountId}:
{emailTopicName}";
        alarmActions.Add(snsAlarmAction);
        return alarmActions;
    }

    /// <summary>
    /// Describe the current alarms, optionally filtered by state.
    /// </summary>
    /// <param name="stateValue">Optional filter for alarm state.</param>
    /// <returns>The list of alarm data.</returns>
    public async Task<List<MetricAlarm>> DescribeAlarms(StateValue? stateValue =
 null)
    {
        List<MetricAlarm> alarms = new List<MetricAlarm>();
        var paginatedDescribeAlarms =
 _amazonCloudWatch.Paginators.DescribeAlarms(
            new DescribeAlarmsRequest()
            {
                StateValue = stateValue
            });

        await foreach (var data in paginatedDescribeAlarms.MetricAlarms)
        {
            alarms.Add(data);
        }
        return alarms;
    }

    /// <summary>
    /// Describe the current alarms for a specific metric.
    /// </summary>
    /// <param name="metricNamespace">The namespace of the metric.</param>
    /// <param name="metricName">The name of the metric.</param>
    /// <returns>The list of alarm data.</returns>
    public async Task<List<MetricAlarm>> DescribeAlarmsForMetric(string
 metricNamespace, string metricName)
    {
        var alarmsResult = await _amazonCloudWatch.DescribeAlarmsForMetricAsync(
            new DescribeAlarmsForMetricRequest()
            {
                Namespace = metricNamespace,
```

```
                MetricName = metricName
            });

        return alarmsResult.MetricAlarms;
    }

    /// <summary>
    /// Describe the history of an alarm for a number of days in the past.
    /// </summary>
    /// <param name="alarmName">The name of the alarm.</param>
    /// <param name="historyDays">The number of days in the past.</param>
    /// <returns>The list of alarm history data.</returns>
    public async Task<List<AlarmHistoryItem>> DescribeAlarmHistory(string
alarmName, int historyDays)
    {
        List<AlarmHistoryItem> alarmHistory = new List<AlarmHistoryItem>();
        var paginatedAlarmHistory =
_amazonCloudWatch.Paginators.DescribeAlarmHistory(
            new DescribeAlarmHistoryRequest()
            {
                AlarmName = alarmName,
                EndDateUtc = DateTime.UtcNow,
                HistoryItemType = HistoryItemType.StateUpdate,
                StartDateUtc = DateTime.UtcNow.AddDays(-historyDays)
            });

        await foreach (var data in paginatedAlarmHistory.AlarmHistoryItems)
        {
            alarmHistory.Add(data);
        }
        return alarmHistory;
    }

    /// <summary>
    /// Delete a list of alarms from CloudWatch.
    /// </summary>
    /// <param name="alarmNames">A list of names of alarms to delete.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DeleteAlarms(List<string> alarmNames)
    {
        var deleteAlarmsResult = await _amazonCloudWatch.DeleteAlarmsAsync(
            new DeleteAlarmsRequest()
            {
                AlarmNames = alarmNames
```

```
            });

        return deleteAlarmsResult.HttpStatusCode == HttpStatusCode.OK;
    }

    /// <summary>
    /// Disable the actions for a list of alarms from CloudWatch.
    /// </summary>
    /// <param name="alarmNames">A list of names of alarms.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DisableAlarmActions(List<string> alarmNames)
    {
        var disableAlarmActionsResult = await
_amazonCloudWatch.DisableAlarmActionsAsync(
            new DisableAlarmActionsRequest()
            {
                AlarmNames = alarmNames
            });

        return disableAlarmActionsResult.HttpStatusCode == HttpStatusCode.OK;
    }

    /// <summary>
    /// Enable the actions for a list of alarms from CloudWatch.
    /// </summary>
    /// <param name="alarmNames">A list of names of alarms.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> EnableAlarmActions(List<string> alarmNames)
    {
        var enableAlarmActionsResult = await
_amazonCloudWatch.EnableAlarmActionsAsync(
            new EnableAlarmActionsRequest()
            {
                AlarmNames = alarmNames
            });

        return enableAlarmActionsResult.HttpStatusCode == HttpStatusCode.OK;
    }

    /// <summary>
    /// Add an anomaly detector for a single metric.
    /// </summary>
    /// <param name="anomalyDetector">A single metric anomaly detector.</param>
    /// <returns>True if successful.</returns>
```

```
   public async Task<bool> PutAnomalyDetector(SingleMetricAnomalyDetector
anomalyDetector)
   {
       var putAlarmDetectorResult = await
_amazonCloudWatch.PutAnomalyDetectorAsync(
           new PutAnomalyDetectorRequest()
           {
               SingleMetricAnomalyDetector = anomalyDetector
           });

       return putAlarmDetectorResult.HttpStatusCode == HttpStatusCode.OK;
   }

   /// <summary>
   /// Describe anomaly detectors for a metric and namespace.
   /// </summary>
   /// <param name="metricNamespace">The namespace of the metric.</param>
   /// <param name="metricName">The metric of the anomaly detectors.</param>
   /// <returns>The list of detectors.</returns>
   public async Task<List<AnomalyDetector>> DescribeAnomalyDetectors(string
metricNamespace, string metricName)
   {
       List<AnomalyDetector> detectors = new List<AnomalyDetector>();
       var paginatedDescribeAnomalyDetectors =
_amazonCloudWatch.Paginators.DescribeAnomalyDetectors(
           new DescribeAnomalyDetectorsRequest()
           {
               MetricName = metricName,
               Namespace = metricNamespace
           });

       await foreach (var data in
paginatedDescribeAnomalyDetectors.AnomalyDetectors)
       {
           detectors.Add(data);
       }

       return detectors;
   }

   /// <summary>
   /// Delete a single metric anomaly detector.
   /// </summary>
   /// <param name="anomalyDetector">The anomaly detector to delete.</param>
```

```
    /// <returns>True if successful.</returns>
    public async Task<bool> DeleteAnomalyDetector(SingleMetricAnomalyDetector
 anomalyDetector)
    {
        var deleteAnomalyDetectorResponse = await
 _amazonCloudWatch.DeleteAnomalyDetectorAsync(
            new DeleteAnomalyDetectorRequest()
            {
                SingleMetricAnomalyDetector = anomalyDetector
            });

        return deleteAnomalyDetectorResponse.HttpStatusCode == HttpStatusCode.OK;
    }

    /// <summary>
    /// Delete a list of CloudWatch dashboards.
    /// </summary>
    /// <param name="dashboardNames">List of dashboard names to delete.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DeleteDashboards(List<string> dashboardNames)
    {
        var deleteDashboardsResponse = await
 _amazonCloudWatch.DeleteDashboardsAsync(
            new DeleteDashboardsRequest()
            {
                DashboardNames = dashboardNames
            });

        return deleteDashboardsResponse.HttpStatusCode == HttpStatusCode.OK;
    }
}
```

- API の詳細については、「AWS SDK for .NET API リファレンス」の以下のトピックを参照してください。
  - DeleteAlarms
  - DeleteAnomalyDetector
  - DeleteDashboards
  - DescribeAlarmHistory
  - DescribeAlarms
  - DescribeAlarmsForMetric

- DescribeAnomalyDetectors
- GetMetricData
- GetMetricStatistics
- GetMetricWidgetImage
- ListMetrics
- PutAnomalyDetector
- PutDashboard
- PutMetricAlarm
- PutMetricData

Java

SDK for Java 2.x

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
import com.fasterxml.jackson.core.JsonFactory;
import com.fasterxml.jackson.core.JsonParser;
import com.fasterxml.jackson.databind.ObjectMapper;
import software.amazon.awssdk.auth.credentials.ProfileCredentialsProvider;
import software.amazon.awssdk.core.SdkBytes;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.cloudwatch.CloudWatchClient;
import software.amazon.awssdk.services.cloudwatch.model.AlarmHistoryItem;
import software.amazon.awssdk.services.cloudwatch.model.AlarmType;
import software.amazon.awssdk.services.cloudwatch.model.AnomalyDetector;
import software.amazon.awssdk.services.cloudwatch.model.CloudWatchException;
import software.amazon.awssdk.services.cloudwatch.model.ComparisonOperator;
import
 software.amazon.awssdk.services.cloudwatch.model.DashboardValidationMessage;
import software.amazon.awssdk.services.cloudwatch.model.Datapoint;
import software.amazon.awssdk.services.cloudwatch.model.DeleteAlarmsRequest;
import
 software.amazon.awssdk.services.cloudwatch.model.DeleteAnomalyDetectorRequest;
```

```
import software.amazon.awssdk.services.cloudwatch.model.DeleteDashboardsRequest;
import
 software.amazon.awssdk.services.cloudwatch.model.DescribeAlarmHistoryRequest;
import
 software.amazon.awssdk.services.cloudwatch.model.DescribeAlarmHistoryResponse;
import
 software.amazon.awssdk.services.cloudwatch.model.DescribeAlarmsForMetricRequest;
import
 software.amazon.awssdk.services.cloudwatch.model.DescribeAlarmsForMetricResponse;
import software.amazon.awssdk.services.cloudwatch.model.DescribeAlarmsRequest;
import software.amazon.awssdk.services.cloudwatch.model.DescribeAlarmsResponse;
import
 software.amazon.awssdk.services.cloudwatch.model.DescribeAnomalyDetectorsRequest;
import
 software.amazon.awssdk.services.cloudwatch.model.DescribeAnomalyDetectorsResponse;
import software.amazon.awssdk.services.cloudwatch.model.Dimension;
import software.amazon.awssdk.services.cloudwatch.model.GetMetricDataRequest;
import software.amazon.awssdk.services.cloudwatch.model.GetMetricDataResponse;
import
 software.amazon.awssdk.services.cloudwatch.model.GetMetricStatisticsRequest;
import
 software.amazon.awssdk.services.cloudwatch.model.GetMetricStatisticsResponse;
import
 software.amazon.awssdk.services.cloudwatch.model.GetMetricWidgetImageRequest;
import
 software.amazon.awssdk.services.cloudwatch.model.GetMetricWidgetImageResponse;
import software.amazon.awssdk.services.cloudwatch.model.HistoryItemType;
import software.amazon.awssdk.services.cloudwatch.model.ListMetricsRequest;
import software.amazon.awssdk.services.cloudwatch.model.ListMetricsResponse;
import software.amazon.awssdk.services.cloudwatch.model.Metric;
import software.amazon.awssdk.services.cloudwatch.model.MetricAlarm;
import software.amazon.awssdk.services.cloudwatch.model.MetricDataQuery;
import software.amazon.awssdk.services.cloudwatch.model.MetricDataResult;
import software.amazon.awssdk.services.cloudwatch.model.MetricDatum;
import software.amazon.awssdk.services.cloudwatch.model.MetricStat;
import
 software.amazon.awssdk.services.cloudwatch.model.PutAnomalyDetectorRequest;
import software.amazon.awssdk.services.cloudwatch.model.PutDashboardRequest;
import software.amazon.awssdk.services.cloudwatch.model.PutDashboardResponse;
import software.amazon.awssdk.services.cloudwatch.model.PutMetricAlarmRequest;
import software.amazon.awssdk.services.cloudwatch.model.PutMetricDataRequest;
import software.amazon.awssdk.services.cloudwatch.model.ScanBy;
import
 software.amazon.awssdk.services.cloudwatch.model.SingleMetricAnomalyDetector;
```

```
import software.amazon.awssdk.services.cloudwatch.model.StandardUnit;
import software.amazon.awssdk.services.cloudwatch.model.Statistic;
import
 software.amazon.awssdk.services.cloudwatch.paginators.ListDashboardsIterable;
import software.amazon.awssdk.services.cloudwatch.paginators.ListMetricsIterable;
import java.io.BufferedReader;
import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStreamReader;
import java.nio.file.Files;
import java.nio.file.Paths;
import java.time.Instant;
import java.time.ZoneOffset;
import java.time.ZonedDateTime;
import java.time.format.DateTimeFormatter;
import java.time.temporal.ChronoUnit;
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 *
 * To enable billing metrics and statistics for this example, make sure billing
 * alerts are enabled for your account:
 * https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/
monitor_estimated_charges_with_cloudwatch.html#turning_on_billing_metrics
 *
 * This Java code example performs the following tasks:
 *
 * 1. List available namespaces from Amazon CloudWatch.
 * 2. List available metrics within the selected Namespace.
 * 3. Get statistics for the selected metric over the last day.
 * 4. Get CloudWatch estimated billing for the last week.
 * 5. Create a new CloudWatch dashboard with metrics.
 * 6. List dashboards using a paginator.
 * 7. Create a new custom metric by adding data for it.
```

```
 * 8. Add the custom metric to the dashboard.
 * 9. Create an alarm for the custom metric.
 * 10. Describe current alarms.
 * 11. Get current data for the new custom metric.
 * 12. Push data into the custom metric to trigger the alarm.
 * 13. Check the alarm state using the action DescribeAlarmsForMetric.
 * 14. Get alarm history for the new alarm.
 * 15. Add an anomaly detector for the custom metric.
 * 16. Describe current anomaly detectors.
 * 17. Get a metric image for the custom metric.
 * 18. Clean up the Amazon CloudWatch resources.
 */
public class CloudWatchScenario {
    public static final String DASHES = new String(new char[80]).replace("\0",
 "-");

    public static void main(String[] args) throws IOException {
        final String usage = """

                Usage:
                  <myDate> <costDateWeek> <dashboardName> <dashboardJson>
 <dashboardAdd> <settings> <metricImage> \s

                Where:
                  myDate - The start date to use to get metric statistics. (For
 example, 2023-01-11T18:35:24.00Z.)\s
                  costDateWeek - The start date to use to get AWS/Billinget
 statistics. (For example, 2023-01-11T18:35:24.00Z.)\s
                  dashboardName - The name of the dashboard to create.\s
                  dashboardJson - The location of a JSON file to use to create a
 dashboard. (See Readme file.)\s
                  dashboardAdd - The location of a JSON file to use to update a
 dashboard. (See Readme file.)\s
                  settings - The location of a JSON file from which various
 values are read. (See Readme file.)\s
                  metricImage - The location of a BMP file that is used to create
 a graph.\s
                """;

        if (args.length != 7) {
            System.out.println(usage);
            System.exit(1);
        }
```

```
        Region region = Region.US_EAST_1;
        String myDate = args[0];
        String costDateWeek = args[1];
        String dashboardName = args[2];
        String dashboardJson = args[3];
        String dashboardAdd = args[4];
        String settings = args[5];
        String metricImage = args[6];

        Double dataPoint = Double.parseDouble("10.0");
        Scanner sc = new Scanner(System.in);
        CloudWatchClient cw = CloudWatchClient.builder()
                .region(region)
                .credentialsProvider(ProfileCredentialsProvider.create())
                .build();

        System.out.println(DASHES);
        System.out.println("Welcome to the Amazon CloudWatch example scenario.");
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println(
                "1. List at least five available unique namespaces from Amazon
CloudWatch. Select one from the list.");
        ArrayList<String> list = listNameSpaces(cw);
        for (int z = 0; z < 5; z++) {
            int index = z + 1;
            System.out.println("    " + index + ". " + list.get(z));
        }

        String selectedNamespace = "";
        String selectedMetrics = "";
        int num = Integer.parseInt(sc.nextLine());
        if (1 <= num && num <= 5) {
            selectedNamespace = list.get(num - 1);
        } else {
            System.out.println("You did not select a valid option.");
            System.exit(1);
        }
        System.out.println("You selected " + selectedNamespace);
        System.out.println(DASHES);

        System.out.println(DASHES);
```

```
        System.out.println("2. List available metrics within the selected
namespace and select one from the list.");
        ArrayList<String> metList = listMets(cw, selectedNamespace);
        for (int z = 0; z < 5; z++) {
            int index = z + 1;
            System.out.println("    " + index + ". " + metList.get(z));
        }
        num = Integer.parseInt(sc.nextLine());
        if (1 <= num && num <= 5) {
            selectedMetrics = metList.get(num - 1);
        } else {
            System.out.println("You did not select a valid option.");
            System.exit(1);
        }
        System.out.println("You selected " + selectedMetrics);
        Dimension myDimension = getSpecificMet(cw, selectedNamespace);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("3. Get statistics for the selected metric over the
last day.");
        String metricOption = "";
        ArrayList<String> statTypes = new ArrayList<>();
        statTypes.add("SampleCount");
        statTypes.add("Average");
        statTypes.add("Sum");
        statTypes.add("Minimum");
        statTypes.add("Maximum");

        for (int t = 0; t < 5; t++) {
            System.out.println("    " + (t + 1) + ". " + statTypes.get(t));
        }
        System.out.println("Select a metric statistic by entering a number from
the preceding list:");
        num = Integer.parseInt(sc.nextLine());
        if (1 <= num && num <= 5) {
            metricOption = statTypes.get(num - 1);
        } else {
            System.out.println("You did not select a valid option.");
            System.exit(1);
        }
        System.out.println("You selected " + metricOption);
        getAndDisplayMetricStatistics(cw, selectedNamespace, selectedMetrics,
metricOption, myDate, myDimension);
```

```
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("4. Get CloudWatch estimated billing for the last
week.");
        getMetricStatistics(cw, costDateWeek);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("5. Create a new CloudWatch dashboard with metrics.");
        createDashboardWithMetrics(cw, dashboardName, dashboardJson);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("6. List dashboards using a paginator.");
        listDashboards(cw);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("7. Create a new custom metric by adding data to
it.");
        createNewCustomMetric(cw, dataPoint);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("8. Add an additional metric to the dashboard.");
        addMetricToDashboard(cw, dashboardAdd, dashboardName);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("9. Create an alarm for the custom metric.");
        String alarmName = createAlarm(cw, settings);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("10. Describe ten current alarms.");
        describeAlarms(cw);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("11. Get current data for new custom metric.");
        getCustomMetricData(cw, settings);
        System.out.println(DASHES);
```

```
        System.out.println(DASHES);
        System.out.println("12. Push data into the custom metric to trigger the
alarm.");
        addMetricDataForAlarm(cw, settings);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("13. Check the alarm state using the action
DescribeAlarmsForMetric.");
        checkForMetricAlarm(cw, settings);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("14. Get alarm history for the new alarm.");
        getAlarmHistory(cw, settings, myDate);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("15. Add an anomaly detector for the custom metric.");
        addAnomalyDetector(cw, settings);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("16. Describe current anomaly detectors.");
        describeAnomalyDetectors(cw, settings);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("17. Get a metric image for the custom metric.");
        getAndOpenMetricImage(cw, metricImage);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("18. Clean up the Amazon CloudWatch resources.");
        deleteDashboard(cw, dashboardName);
        deleteCWAlarm(cw, alarmName);
        deleteAnomalyDetector(cw, settings);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("The Amazon CloudWatch example scenario is
complete.");
        System.out.println(DASHES);
        cw.close();
```

```
    }

    public static void deleteAnomalyDetector(CloudWatchClient cw, String
fileName) {
        try {
            // Read values from the JSON file.
            JsonParser parser = new JsonFactory().createParser(new
File(fileName));
            com.fasterxml.jackson.databind.JsonNode rootNode = new
ObjectMapper().readTree(parser);
            String customMetricNamespace =
rootNode.findValue("customMetricNamespace").asText();
            String customMetricName =
rootNode.findValue("customMetricName").asText();

            SingleMetricAnomalyDetector singleMetricAnomalyDetector =
SingleMetricAnomalyDetector.builder()
                    .metricName(customMetricName)
                    .namespace(customMetricNamespace)
                    .stat("Maximum")
                    .build();

            DeleteAnomalyDetectorRequest request =
DeleteAnomalyDetectorRequest.builder()
                    .singleMetricAnomalyDetector(singleMetricAnomalyDetector)
                    .build();

            cw.deleteAnomalyDetector(request);
            System.out.println("Successfully deleted the Anomaly Detector.");

        } catch (CloudWatchException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    public static void deleteCWAlarm(CloudWatchClient cw, String alarmName) {
        try {
            DeleteAlarmsRequest request = DeleteAlarmsRequest.builder()
                    .alarmNames(alarmName)
                    .build();
```

```
            cw.deleteAlarms(request);
            System.out.println("Successfully deleted alarm " + alarmName);

        } catch (CloudWatchException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }

    public static void deleteDashboard(CloudWatchClient cw, String dashboardName)
{
        try {
            DeleteDashboardsRequest dashboardsRequest =
DeleteDashboardsRequest.builder()
                    .dashboardNames(dashboardName)
                    .build();
            cw.deleteDashboards(dashboardsRequest);
            System.out.println(dashboardName + " was successfully deleted.");

        } catch (CloudWatchException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }

    public static void getAndOpenMetricImage(CloudWatchClient cw, String
fileName) {
        System.out.println("Getting Image data for custom metric.");
        try {
            String myJSON = "{\n" +
                    "  \"title\": \"Example Metric Graph\",\n" +
                    "  \"view\": \"timeSeries\",\n" +
                    "  \"stacked \": false,\n" +
                    "  \"period\": 10,\n" +
                    "  \"width\": 1400,\n" +
                    "  \"height\": 600,\n" +
                    "  \"metrics\": [\n" +
                    "    [\n" +
                    "      \"AWS/Billing\",\n" +
                    "      \"EstimatedCharges\",\n" +
                    "      \"Currency\",\n" +
                    "      \"USD\"\n" +
                    "    ]\n" +
                    "  ]\n" +
```

```
                    "}";

            GetMetricWidgetImageRequest imageRequest =
GetMetricWidgetImageRequest.builder()
                    .metricWidget(myJSON)
                    .build();

            GetMetricWidgetImageResponse response =
cw.getMetricWidgetImage(imageRequest);
            SdkBytes sdkBytes = response.metricWidgetImage();
            byte[] bytes = sdkBytes.asByteArray();
            File outputFile = new File(fileName);
            try (FileOutputStream outputStream = new
FileOutputStream(outputFile)) {
                outputStream.write(bytes);
            }

        } catch (CloudWatchException | IOException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }

    public static void describeAnomalyDetectors(CloudWatchClient cw, String
fileName) {
        try {
            // Read values from the JSON file.
            JsonParser parser = new JsonFactory().createParser(new
File(fileName));
            com.fasterxml.jackson.databind.JsonNode rootNode = new
ObjectMapper().readTree(parser);
            String customMetricNamespace =
rootNode.findValue("customMetricNamespace").asText();
            String customMetricName =
rootNode.findValue("customMetricName").asText();
            DescribeAnomalyDetectorsRequest detectorsRequest =
DescribeAnomalyDetectorsRequest.builder()
                    .maxResults(10)
                    .metricName(customMetricName)
                    .namespace(customMetricNamespace)
                    .build();

            DescribeAnomalyDetectorsResponse response =
cw.describeAnomalyDetectors(detectorsRequest);
```

```
            List<AnomalyDetector> anomalyDetectorList =
response.anomalyDetectors();
            for (AnomalyDetector detector : anomalyDetectorList) {
                System.out.println("Metric name: " +
detector.singleMetricAnomalyDetector().metricName());
                System.out.println("State: " + detector.stateValue());
            }

        } catch (CloudWatchException | IOException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }

    public static void addAnomalyDetector(CloudWatchClient cw, String fileName) {
        try {
            // Read values from the JSON file.
            JsonParser parser = new JsonFactory().createParser(new
File(fileName));
            com.fasterxml.jackson.databind.JsonNode rootNode = new
ObjectMapper().readTree(parser);
            String customMetricNamespace =
rootNode.findValue("customMetricNamespace").asText();
            String customMetricName =
rootNode.findValue("customMetricName").asText();

            SingleMetricAnomalyDetector singleMetricAnomalyDetector =
SingleMetricAnomalyDetector.builder()
                    .metricName(customMetricName)
                    .namespace(customMetricNamespace)
                    .stat("Maximum")
                    .build();

            PutAnomalyDetectorRequest anomalyDetectorRequest =
PutAnomalyDetectorRequest.builder()
                    .singleMetricAnomalyDetector(singleMetricAnomalyDetector)
                    .build();

            cw.putAnomalyDetector(anomalyDetectorRequest);
            System.out.println("Added anomaly detector for metric " +
customMetricName + ".");

        } catch (CloudWatchException | IOException e) {
            System.err.println(e.getMessage());
```

```
            System.exit(1);
        }
    }

    public static void getAlarmHistory(CloudWatchClient cw, String fileName,
String date) {
        try {
            // Read values from the JSON file.
            JsonParser parser = new JsonFactory().createParser(new
File(fileName));
            com.fasterxml.jackson.databind.JsonNode rootNode = new
ObjectMapper().readTree(parser);
            String alarmName = rootNode.findValue("exampleAlarmName").asText();

            Instant start = Instant.parse(date);
            Instant endDate = Instant.now();
            DescribeAlarmHistoryRequest historyRequest =
DescribeAlarmHistoryRequest.builder()
                    .startDate(start)
                    .endDate(endDate)
                    .alarmName(alarmName)
                    .historyItemType(HistoryItemType.ACTION)
                    .build();

            DescribeAlarmHistoryResponse response =
cw.describeAlarmHistory(historyRequest);
            List<AlarmHistoryItem> historyItems = response.alarmHistoryItems();
            if (historyItems.isEmpty()) {
                System.out.println("No alarm history data found for " + alarmName
+ ".");
            } else {
                for (AlarmHistoryItem item : historyItems) {
                    System.out.println("History summary: " +
item.historySummary());
                    System.out.println("Time stamp: " + item.timestamp());
                }
            }

        } catch (CloudWatchException | IOException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }
```

```
   public static void checkForMetricAlarm(CloudWatchClient cw, String fileName)
{
       try {
           // Read values from the JSON file.
           JsonParser parser = new JsonFactory().createParser(new
File(fileName));
           com.fasterxml.jackson.databind.JsonNode rootNode = new
ObjectMapper().readTree(parser);
           String customMetricNamespace =
rootNode.findValue("customMetricNamespace").asText();
           String customMetricName =
rootNode.findValue("customMetricName").asText();
           boolean hasAlarm = false;
           int retries = 10;

           DescribeAlarmsForMetricRequest metricRequest =
DescribeAlarmsForMetricRequest.builder()
                   .metricName(customMetricName)
                   .namespace(customMetricNamespace)
                   .build();

           while (!hasAlarm && retries > 0) {
               DescribeAlarmsForMetricResponse response =
cw.describeAlarmsForMetric(metricRequest);
               hasAlarm = response.hasMetricAlarms();
               retries--;
               Thread.sleep(20000);
               System.out.println(".");
           }
           if (!hasAlarm)
               System.out.println("No Alarm state found for " + customMetricName
+ " after 10 retries.");
           else
               System.out.println("Alarm state found for " + customMetricName +
".");

       } catch (CloudWatchException | IOException | InterruptedException e) {
           System.err.println(e.getMessage());
           System.exit(1);
       }
   }

   public static void addMetricDataForAlarm(CloudWatchClient cw, String
fileName) {
```

```
        try {
            // Read values from the JSON file.
            JsonParser parser = new JsonFactory().createParser(new
File(fileName));
            com.fasterxml.jackson.databind.JsonNode rootNode = new
ObjectMapper().readTree(parser);
            String customMetricNamespace =
rootNode.findValue("customMetricNamespace").asText();
            String customMetricName =
rootNode.findValue("customMetricName").asText();

            // Set an Instant object.
            String time =
ZonedDateTime.now(ZoneOffset.UTC).format(DateTimeFormatter.ISO_INSTANT);
            Instant instant = Instant.parse(time);

            MetricDatum datum = MetricDatum.builder()
                    .metricName(customMetricName)
                    .unit(StandardUnit.NONE)
                    .value(1001.00)
                    .timestamp(instant)
                    .build();

            MetricDatum datum2 = MetricDatum.builder()
                    .metricName(customMetricName)
                    .unit(StandardUnit.NONE)
                    .value(1002.00)
                    .timestamp(instant)
                    .build();

            List<MetricDatum> metricDataList = new ArrayList<>();
            metricDataList.add(datum);
            metricDataList.add(datum2);

            PutMetricDataRequest request = PutMetricDataRequest.builder()
                    .namespace(customMetricNamespace)
                    .metricData(metricDataList)
                    .build();

            cw.putMetricData(request);
            System.out.println("Added metric values for for metric " +
customMetricName);

        } catch (CloudWatchException | IOException e) {
```

```
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }

    public static void getCustomMetricData(CloudWatchClient cw, String fileName)
{
        try {
            // Read values from the JSON file.
            JsonParser parser = new JsonFactory().createParser(new
File(fileName));
            com.fasterxml.jackson.databind.JsonNode rootNode = new
ObjectMapper().readTree(parser);
            String customMetricNamespace =
rootNode.findValue("customMetricNamespace").asText();
            String customMetricName =
rootNode.findValue("customMetricName").asText();

            // Set the date.
            Instant nowDate = Instant.now();

            long hours = 1;
            long minutes = 30;
            Instant date2 = nowDate.plus(hours, ChronoUnit.HOURS).plus(minutes,
                    ChronoUnit.MINUTES);

            Metric met = Metric.builder()
                    .metricName(customMetricName)
                    .namespace(customMetricNamespace)
                    .build();

            MetricStat metStat = MetricStat.builder()
                    .stat("Maximum")
                    .period(1)
                    .metric(met)
                    .build();

            MetricDataQuery dataQUery = MetricDataQuery.builder()
                    .metricStat(metStat)
                    .id("foo2")
                    .returnData(true)
                    .build();

            List<MetricDataQuery> dq = new ArrayList<>();
```

```
            dq.add(dataQUery);

            GetMetricDataRequest getMetReq = GetMetricDataRequest.builder()
                    .maxDatapoints(10)
                    .scanBy(ScanBy.TIMESTAMP_DESCENDING)
                    .startTime(nowDate)
                    .endTime(date2)
                    .metricDataQueries(dq)
                    .build();

            GetMetricDataResponse response = cw.getMetricData(getMetReq);
            List<MetricDataResult> data = response.metricDataResults();
            for (MetricDataResult item : data) {
                System.out.println("The label is " + item.label());
                System.out.println("The status code is " +
item.statusCode().toString());
            }

        } catch (CloudWatchException | IOException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }

    public static void describeAlarms(CloudWatchClient cw) {
        try {
            List<AlarmType> typeList = new ArrayList<>();
            typeList.add(AlarmType.METRIC_ALARM);

            DescribeAlarmsRequest alarmsRequest = DescribeAlarmsRequest.builder()
                    .alarmTypes(typeList)
                    .maxRecords(10)
                    .build();

            DescribeAlarmsResponse response = cw.describeAlarms(alarmsRequest);
            List<MetricAlarm> alarmList = response.metricAlarms();
            for (MetricAlarm alarm : alarmList) {
                System.out.println("Alarm name: " + alarm.alarmName());
                System.out.println("Alarm description: " +
alarm.alarmDescription());
            }
        } catch (CloudWatchException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
```

```
        }
    }

    public static String createAlarm(CloudWatchClient cw, String fileName) {
        try {
            // Read values from the JSON file.
            JsonParser parser = new JsonFactory().createParser(new
File(fileName));
            com.fasterxml.jackson.databind.JsonNode rootNode = new
ObjectMapper().readTree(parser);
            String customMetricNamespace =
rootNode.findValue("customMetricNamespace").asText();
            String customMetricName =
rootNode.findValue("customMetricName").asText();
            String alarmName = rootNode.findValue("exampleAlarmName").asText();
            String emailTopic = rootNode.findValue("emailTopic").asText();
            String accountId = rootNode.findValue("accountId").asText();
            String region = rootNode.findValue("region").asText();

            // Create a List for alarm actions.
            List<String> alarmActions = new ArrayList<>();
            alarmActions.add("arn:aws:sns:" + region + ":" + accountId + ":" +
emailTopic);
            PutMetricAlarmRequest alarmRequest = PutMetricAlarmRequest.builder()
                    .alarmActions(alarmActions)
                    .alarmDescription("Example metric alarm")
                    .alarmName(alarmName)

.comparisonOperator(ComparisonOperator.GREATER_THAN_OR_EQUAL_TO_THRESHOLD)
                    .threshold(100.00)
                    .metricName(customMetricName)
                    .namespace(customMetricNamespace)
                    .evaluationPeriods(1)
                    .period(10)
                    .statistic("Maximum")
                    .datapointsToAlarm(1)
                    .treatMissingData("ignore")
                    .build();

            cw.putMetricAlarm(alarmRequest);
            System.out.println(alarmName + " was successfully created!");
            return alarmName;

        } catch (CloudWatchException | IOException e) {
```

```
            System.err.println(e.getMessage());
            System.exit(1);
        }
        return "";
    }

    public static void addMetricToDashboard(CloudWatchClient cw, String fileName,
String dashboardName) {
        try {
            PutDashboardRequest dashboardRequest = PutDashboardRequest.builder()
                    .dashboardName(dashboardName)
                    .dashboardBody(readFileAsString(fileName))
                    .build();

            cw.putDashboard(dashboardRequest);
            System.out.println(dashboardName + " was successfully updated.");

        } catch (CloudWatchException | IOException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }

    public static void createNewCustomMetric(CloudWatchClient cw, Double
dataPoint) {
        try {
            Dimension dimension = Dimension.builder()
                    .name("UNIQUE_PAGES")
                    .value("URLS")
                    .build();

            // Set an Instant object.
            String time =
ZonedDateTime.now(ZoneOffset.UTC).format(DateTimeFormatter.ISO_INSTANT);
            Instant instant = Instant.parse(time);

            MetricDatum datum = MetricDatum.builder()
                    .metricName("PAGES_VISITED")
                    .unit(StandardUnit.NONE)
                    .value(dataPoint)
                    .timestamp(instant)
                    .dimensions(dimension)
                    .build();
```

```
            PutMetricDataRequest request = PutMetricDataRequest.builder()
                    .namespace("SITE/TRAFFIC")
                    .metricData(datum)
                    .build();

            cw.putMetricData(request);
            System.out.println("Added metric values for for metric
PAGES_VISITED");

        } catch (CloudWatchException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }

    public static void listDashboards(CloudWatchClient cw) {
        try {
            ListDashboardsIterable listRes = cw.listDashboardsPaginator();
            listRes.stream()
                    .flatMap(r -> r.dashboardEntries().stream())
                    .forEach(entry -> {
                        System.out.println("Dashboard name is: " +
entry.dashboardName());
                        System.out.println("Dashboard ARN is: " +
entry.dashboardArn());
                    });

        } catch (CloudWatchException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }

    public static void createDashboardWithMetrics(CloudWatchClient cw, String
dashboardName, String fileName) {
        try {
            PutDashboardRequest dashboardRequest = PutDashboardRequest.builder()
                    .dashboardName(dashboardName)
                    .dashboardBody(readFileAsString(fileName))
                    .build();

            PutDashboardResponse response = cw.putDashboard(dashboardRequest);
            System.out.println(dashboardName + " was successfully created.");
```

```
            List<DashboardValidationMessage> messages =
response.dashboardValidationMessages();
            if (messages.isEmpty()) {
                System.out.println("There are no messages in the new Dashboard");
            } else {
                for (DashboardValidationMessage message : messages) {
                    System.out.println("Message is: " + message.message());
                }
            }

        } catch (CloudWatchException | IOException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }

    public static String readFileAsString(String file) throws IOException {
        return new String(Files.readAllBytes(Paths.get(file)));
    }

    public static void getMetricStatistics(CloudWatchClient cw, String
costDateWeek) {
        try {
            Instant start = Instant.parse(costDateWeek);
            Instant endDate = Instant.now();
            Dimension dimension = Dimension.builder()
                    .name("Currency")
                    .value("USD")
                    .build();

            List<Dimension> dimensionList = new ArrayList<>();
            dimensionList.add(dimension);
            GetMetricStatisticsRequest statisticsRequest =
GetMetricStatisticsRequest.builder()
                    .metricName("EstimatedCharges")
                    .namespace("AWS/Billing")
                    .dimensions(dimensionList)
                    .statistics(Statistic.MAXIMUM)
                    .startTime(start)
                    .endTime(endDate)
                    .period(86400)
                    .build();
```

```
            GetMetricStatisticsResponse response =
cw.getMetricStatistics(statisticsRequest);
            List<Datapoint> data = response.datapoints();
            if (!data.isEmpty()) {
                for (Datapoint datapoint : data) {
                    System.out
                            .println("Timestamp: " + datapoint.timestamp() + "
Maximum value: " + datapoint.maximum());
                }
            } else {
                System.out.println("The returned data list is empty");
            }

        } catch (CloudWatchException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }

    public static void getAndDisplayMetricStatistics(CloudWatchClient cw, String
nameSpace, String metVal,
            String metricOption, String date, Dimension myDimension) {
        try {
            Instant start = Instant.parse(date);
            Instant endDate = Instant.now();

            GetMetricStatisticsRequest statisticsRequest =
GetMetricStatisticsRequest.builder()
                    .endTime(endDate)
                    .startTime(start)
                    .dimensions(myDimension)
                    .metricName(metVal)
                    .namespace(nameSpace)
                    .period(86400)
                    .statistics(Statistic.fromValue(metricOption))
                    .build();

            GetMetricStatisticsResponse response =
cw.getMetricStatistics(statisticsRequest);
            List<Datapoint> data = response.datapoints();
            if (!data.isEmpty()) {
                for (Datapoint datapoint : data) {
                    System.out
```

```
                            .println("Timestamp: " + datapoint.timestamp() + "
Maximum value: " + datapoint.maximum());
                }
            } else {
                System.out.println("The returned data list is empty");
            }

        } catch (CloudWatchException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }

    public static Dimension getSpecificMet(CloudWatchClient cw, String namespace)
{
        try {
            ListMetricsRequest request = ListMetricsRequest.builder()
                    .namespace(namespace)
                    .build();

            ListMetricsResponse response = cw.listMetrics(request);
            List<Metric> myList = response.metrics();
            Metric metric = myList.get(0);
            return metric.dimensions().get(0);

        } catch (CloudWatchException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
        return null;
    }

    public static ArrayList<String> listMets(CloudWatchClient cw, String
namespace) {
        try {
            ArrayList<String> metList = new ArrayList<>();
            ListMetricsRequest request = ListMetricsRequest.builder()
                    .namespace(namespace)
                    .build();

            ListMetricsIterable listRes = cw.listMetricsPaginator(request);
            listRes.stream()
                    .flatMap(r -> r.metrics().stream())
                    .forEach(metrics -> metList.add(metrics.metricName()));
```

```
            return metList;

        } catch (CloudWatchException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
        return null;
    }

    public static ArrayList<String> listNameSpaces(CloudWatchClient cw) {
        try {
            ArrayList<String> nameSpaceList = new ArrayList<>();
            ListMetricsRequest request = ListMetricsRequest.builder()
                    .build();

            ListMetricsIterable listRes = cw.listMetricsPaginator(request);
            listRes.stream()
                    .flatMap(r -> r.metrics().stream())
                    .forEach(metrics -> {
                        String data = metrics.namespace();
                        if (!nameSpaceList.contains(data)) {
                            nameSpaceList.add(data);
                        }
                    });

            return nameSpaceList;
        } catch (CloudWatchException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
        return null;
    }
}
```

- API の詳細については、「AWS SDK for Java 2.x API リファレンス」の以下のトピックを参照してください。
  - DeleteAlarms
  - DeleteAnomalyDetector
  - DeleteDashboards
  - DescribeAlarmHistory

- DescribeAlarms
- DescribeAlarmsForMetric
- DescribeAnomalyDetectors
- GetMetricData
- GetMetricStatistics
- GetMetricWidgetImage
- ListMetrics
- PutAnomalyDetector
- PutDashboard
- PutMetricAlarm
- PutMetricData

Kotlin

SDK for Kotlin

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
/**
 Before running this Kotlin code example, set up your development environment,
 including your credentials.

 For more information, see the following documentation topic:
 https://docs.aws.amazon.com/sdk-for-kotlin/latest/developer-guide/setup.html

 To enable billing metrics and statistics for this example, make sure billing
 alerts are enabled for your account:
 https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/
monitor_estimated_charges_with_cloudwatch.html#turning_on_billing_metrics

 This Kotlin code example performs the following tasks:
```

```
 1. List available namespaces from Amazon CloudWatch. Select a namespace from the
 list.
 2. List available metrics within the selected namespace.
 3. Get statistics for the selected metric over the last day.
 4. Get CloudWatch estimated billing for the last week.
 5. Create a new CloudWatch dashboard with metrics.
 6. List dashboards using a paginator.
 7. Create a new custom metric by adding data for it.
 8. Add the custom metric to the dashboard.
 9. Create an alarm for the custom metric.
 10. Describe current alarms.
 11. Get current data for the new custom metric.
 12. Push data into the custom metric to trigger the alarm.
 13. Check the alarm state using the action DescribeAlarmsForMetric.
 14. Get alarm history for the new alarm.
 15. Add an anomaly detector for the custom metric.
 16. Describe current anomaly detectors.
 17. Get a metric image for the custom metric.
 18. Clean up the Amazon CloudWatch resources.
 */

val DASHES: String? = String(CharArray(80)).replace("\u0000", "-")
suspend fun main(args: Array<String>) {
    val usage = """
        Usage:
            <myDate> <costDateWeek> <dashboardName> <dashboardJson>
 <dashboardAdd> <settings> <metricImage>

        Where:
            myDate - The start date to use to get metric statistics. (For
 example, 2023-01-11T18:35:24.00Z.)
            costDateWeek - The start date to use to get AWS Billing and Cost
 Management statistics. (For example, 2023-01-11T18:35:24.00Z.)
            dashboardName - The name of the dashboard to create.
            dashboardJson - The location of a JSON file to use to create a
 dashboard. (See Readme file.)
            dashboardAdd - The location of a JSON file to use to update a
 dashboard. (See Readme file.)
            settings - The location of a JSON file from which various values are
 read. (See Readme file.)
            metricImage - The location of a BMP file that is used to create a
 graph.
    """
```

```
    if (args.size != 7) {
        println(usage)
        System.exit(1)
    }

    val myDate = args[0]
    val costDateWeek = args[1]
    val dashboardName = args[2]
    val dashboardJson = args[3]
    val dashboardAdd = args[4]
    val settings = args[5]
    var metricImage = args[6]
    val dataPoint = "10.0".toDouble()
    val inOb = Scanner(System.`in`)

    println(DASHES)
    println("Welcome to the Amazon CloudWatch example scenario.")
    println(DASHES)

    println(DASHES)
    println("1. List at least five available unique namespaces from Amazon
CloudWatch. Select a CloudWatch namespace from the list.")
    val list: ArrayList<String> = listNameSpaces()
    for (z in 0..4) {
        println("    ${z + 1}. ${list[z]}")
    }

    var selectedNamespace: String
    var selectedMetrics = ""
    var num = inOb.nextLine().toInt()
    println("You selected $num")

    if (1 <= num && num <= 5) {
        selectedNamespace = list[num - 1]
    } else {
        println("You did not select a valid option.")
        exitProcess(1)
    }
    println("You selected $selectedNamespace")
    println(DASHES)

    println(DASHES)
    println("2. List available metrics within the selected namespace and select
one from the list.")
```

```
    val metList = listMets(selectedNamespace)
    for (z in 0..4) {
        println("    ${ z + 1}. ${metList?.get(z)}")
    }
    num = inOb.nextLine().toInt()
    if (1 <= num && num <= 5) {
        selectedMetrics = metList!![num - 1]
    } else {
        println("You did not select a valid option.")
        System.exit(1)
    }
    println("You selected $selectedMetrics")
    val myDimension = getSpecificMet(selectedNamespace)
    if (myDimension == null) {
        println("Error - Dimension is null")
        exitProcess(1)
    }
    println(DASHES)

    println(DASHES)
    println("3. Get statistics for the selected metric over the last day.")
    val metricOption: String
    val statTypes = ArrayList<String>()
    statTypes.add("SampleCount")
    statTypes.add("Average")
    statTypes.add("Sum")
    statTypes.add("Minimum")
    statTypes.add("Maximum")

    for (t in 0..4) {
        println("    ${t + 1}. ${statTypes[t]}")
    }
    println("Select a metric statistic by entering a number from the preceding
list:")
    num = inOb.nextLine().toInt()
    if (1 <= num && num <= 5) {
        metricOption = statTypes[num - 1]
    } else {
        println("You did not select a valid option.")
        exitProcess(1)
    }
    println("You selected $metricOption")
    getAndDisplayMetricStatistics(selectedNamespace, selectedMetrics,
metricOption, myDate, myDimension)
```

```
    println(DASHES)

    println(DASHES)
    println("4. Get CloudWatch estimated billing for the last week.")
    getMetricStatistics(costDateWeek)
    println(DASHES)

    println(DASHES)
    println("5. Create a new CloudWatch dashboard with metrics.")
    createDashboardWithMetrics(dashboardName, dashboardJson)
    println(DASHES)

    println(DASHES)
    println("6. List dashboards using a paginator.")
    listDashboards()
    println(DASHES)

    println(DASHES)
    println("7. Create a new custom metric by adding data to it.")
    createNewCustomMetric(dataPoint)
    println(DASHES)

    println(DASHES)
    println("8. Add an additional metric to the dashboard.")
    addMetricToDashboard(dashboardAdd, dashboardName)
    println(DASHES)

    println(DASHES)
    println("9. Create an alarm for the custom metric.")
    val alarmName: String = createAlarm(settings)
    println(DASHES)

    println(DASHES)
    println("10. Describe 10 current alarms.")
    describeAlarms()
    println(DASHES)

    println(DASHES)
    println("11. Get current data for the new custom metric.")
    getCustomMetricData(settings)
    println(DASHES)

    println(DASHES)
    println("12. Push data into the custom metric to trigger the alarm.")
```

```
    addMetricDataForAlarm(settings)
    println(DASHES)

    println(DASHES)
    println("13. Check the alarm state using the action
 DescribeAlarmsForMetric.")
    checkForMetricAlarm(settings)
    println(DASHES)

    println(DASHES)
    println("14. Get alarm history for the new alarm.")
    getAlarmHistory(settings, myDate)
    println(DASHES)

    println(DASHES)
    println("15. Add an anomaly detector for the custom metric.")
    addAnomalyDetector(settings)
    println(DASHES)

    println(DASHES)
    println("16. Describe current anomaly detectors.")
    describeAnomalyDetectors(settings)
    println(DASHES)

    println(DASHES)
    println("17. Get a metric image for the custom metric.")
    getAndOpenMetricImage(metricImage)
    println(DASHES)

    println(DASHES)
    println("18. Clean up the Amazon CloudWatch resources.")
    deleteDashboard(dashboardName)
    deleteAlarm(alarmName)
    deleteAnomalyDetector(settings)
    println(DASHES)

    println(DASHES)
    println("The Amazon CloudWatch example scenario is complete.")
    println(DASHES)
}

suspend fun deleteAnomalyDetector(fileName: String) {
    // Read values from the JSON file.
    val parser = JsonFactory().createParser(File(fileName))
```

```
    val rootNode = ObjectMapper().readTree<JsonNode>(parser)
    val customMetricNamespace =
 rootNode.findValue("customMetricNamespace").asText()
    val customMetricName = rootNode.findValue("customMetricName").asText()

    val singleMetricAnomalyDetectorVal = SingleMetricAnomalyDetector {
        metricName = customMetricName
        namespace = customMetricNamespace
        stat = "Maximum"
    }

    val request = DeleteAnomalyDetectorRequest {
        singleMetricAnomalyDetector = singleMetricAnomalyDetectorVal
    }

    CloudWatchClient { region = "us-east-1" }.use { cwClient ->
        cwClient.deleteAnomalyDetector(request)
        println("Successfully deleted the Anomaly Detector.")
    }
}

suspend fun deleteAlarm(alarmNameVal: String) {
    val request = DeleteAlarmsRequest {
        alarmNames = listOf(alarmNameVal)
    }

    CloudWatchClient { region = "us-east-1" }.use { cwClient ->
        cwClient.deleteAlarms(request)
        println("Successfully deleted alarm $alarmNameVal")
    }
}

suspend fun deleteDashboard(dashboardName: String) {
    val dashboardsRequest = DeleteDashboardsRequest {
        dashboardNames = listOf(dashboardName)
    }
    CloudWatchClient { region = "us-east-1" }.use { cwClient ->
        cwClient.deleteDashboards(dashboardsRequest)
        println("$dashboardName was successfully deleted.")
    }
}

suspend fun getAndOpenMetricImage(fileName: String) {
    println("Getting Image data for custom metric.")
```

```
    val myJSON = """{
        "title": "Example Metric Graph",
        "view": "timeSeries",
        "stacked ": false,
        "period": 10,
        "width": 1400,
        "height": 600,
        "metrics": [
            [
            "AWS/Billing",
            "EstimatedCharges",
            "Currency",
            "USD"
            ]
        ]
        }"""

    val imageRequest = GetMetricWidgetImageRequest {
        metricWidget = myJSON
    }

    CloudWatchClient { region = "us-east-1" }.use { cwClient ->
        val response = cwClient.getMetricWidgetImage(imageRequest)
        val bytes = response.metricWidgetImage
        if (bytes != null) {
            File(fileName).writeBytes(bytes)
        }
    }
    println("You have successfully written data to $fileName")
}

suspend fun describeAnomalyDetectors(fileName: String) {
    // Read values from the JSON file.
    val parser = JsonFactory().createParser(File(fileName))
    val rootNode = ObjectMapper().readTree<JsonNode>(parser)
    val customMetricNamespace =
 rootNode.findValue("customMetricNamespace").asText()
    val customMetricName = rootNode.findValue("customMetricName").asText()

    val detectorsRequest = DescribeAnomalyDetectorsRequest {
        maxResults = 10
        metricName = customMetricName
        namespace = customMetricNamespace
    }
```

```
    CloudWatchClient { region = "us-east-1" }.use { cwClient ->
        val response = cwClient.describeAnomalyDetectors(detectorsRequest)
        response.anomalyDetectors?.forEach { detector ->
            println("Metric name:
 ${detector.singleMetricAnomalyDetector?.metricName}")
            println("State: ${detector.stateValue}")
        }
    }
}

suspend fun addAnomalyDetector(fileName: String?) {
    // Read values from the JSON file.
    val parser = JsonFactory().createParser(File(fileName))
    val rootNode = ObjectMapper().readTree<JsonNode>(parser)
    val customMetricNamespace =
 rootNode.findValue("customMetricNamespace").asText()
    val customMetricName = rootNode.findValue("customMetricName").asText()

    val singleMetricAnomalyDetectorVal = SingleMetricAnomalyDetector {
        metricName = customMetricName
        namespace = customMetricNamespace
        stat = "Maximum"
    }

    val anomalyDetectorRequest = PutAnomalyDetectorRequest {
        singleMetricAnomalyDetector = singleMetricAnomalyDetectorVal
    }

    CloudWatchClient { region = "us-east-1" }.use { cwClient ->
        cwClient.putAnomalyDetector(anomalyDetectorRequest)
        println("Added anomaly detector for metric $customMetricName.")
    }
}

suspend fun getAlarmHistory(fileName: String, date: String) {
    // Read values from the JSON file.
    val parser = JsonFactory().createParser(File(fileName))
    val rootNode = ObjectMapper().readTree<JsonNode>(parser)
    val alarmNameVal = rootNode.findValue("exampleAlarmName").asText()
    val start = Instant.parse(date)
    val endDateVal = Instant.now()

    val historyRequest = DescribeAlarmHistoryRequest {
        startDate = aws.smithy.kotlin.runtime.time.Instant(start)
```

```
        endDate = aws.smithy.kotlin.runtime.time.Instant(endDateVal)
        alarmName = alarmNameVal
        historyItemType = HistoryItemType.Action
    }

    CloudWatchClient { credentialsProvider = EnvironmentCredentialsProvider();
 region = "us-east-1" }.use { cwClient ->
        val response = cwClient.describeAlarmHistory(historyRequest)
        val historyItems = response.alarmHistoryItems
        if (historyItems != null) {
            if (historyItems.isEmpty()) {
                println("No alarm history data found for $alarmNameVal.")
            } else {
                for (item in historyItems) {
                    println("History summary ${item.historySummary}")
                    println("Time stamp: ${item.timestamp}")
                }
            }
        }
    }
}

suspend fun checkForMetricAlarm(fileName: String?) {
    // Read values from the JSON file.
    val parser = JsonFactory().createParser(File(fileName))
    val rootNode = ObjectMapper().readTree<JsonNode>(parser)
    val customMetricNamespace =
 rootNode.findValue("customMetricNamespace").asText()
    val customMetricName = rootNode.findValue("customMetricName").asText()
    var hasAlarm = false
    var retries = 10

    val metricRequest = DescribeAlarmsForMetricRequest {
        metricName = customMetricName
        namespace = customMetricNamespace
    }
    CloudWatchClient { region = "us-east-1" }.use { cwClient ->
        while (!hasAlarm && retries > 0) {
            val response = cwClient.describeAlarmsForMetric(metricRequest)
            if (response.metricAlarms?.count()!! > 0) {
                hasAlarm = true
            }
            retries--
            delay(20000)
```

```
            println(".")
        }
        if (!hasAlarm) println("No Alarm state found for $customMetricName after
 10 retries.") else println("Alarm state found for $customMetricName.")
    }
}

suspend fun addMetricDataForAlarm(fileName: String?) {
    // Read values from the JSON file.
    val parser = JsonFactory().createParser(File(fileName))
    val rootNode = ObjectMapper().readTree<JsonNode>(parser)
    val customMetricNamespace =
 rootNode.findValue("customMetricNamespace").asText()
    val customMetricName = rootNode.findValue("customMetricName").asText()

    // Set an Instant object.
    val time =
 ZonedDateTime.now(ZoneOffset.UTC).format(DateTimeFormatter.ISO_INSTANT)
    val instant = Instant.parse(time)
    val datum = MetricDatum {
        metricName = customMetricName
        unit = StandardUnit.None
        value = 1001.00
        timestamp = aws.smithy.kotlin.runtime.time.Instant(instant)
    }

    val datum2 = MetricDatum {
        metricName = customMetricName
        unit = StandardUnit.None
        value = 1002.00
        timestamp = aws.smithy.kotlin.runtime.time.Instant(instant)
    }

    val metricDataList = ArrayList<MetricDatum>()
    metricDataList.add(datum)
    metricDataList.add(datum2)

    val request = PutMetricDataRequest {
        namespace = customMetricNamespace
        metricData = metricDataList
    }

    CloudWatchClient { region = "us-east-1" }.use { cwClient ->
        cwClient.putMetricData(request)
```

```
        println("Added metric values for for metric $customMetricName")
    }
}

suspend fun getCustomMetricData(fileName: String) {
    // Read values from the JSON file.
    val parser = JsonFactory().createParser(File(fileName))
    val rootNode = ObjectMapper().readTree<JsonNode>(parser)
    val customMetricNamespace =
 rootNode.findValue("customMetricNamespace").asText()
    val customMetricName = rootNode.findValue("customMetricName").asText()

    // Set the date.
    val nowDate = Instant.now()
    val hours: Long = 1
    val minutes: Long = 30
    val date2 = nowDate.plus(hours, ChronoUnit.HOURS).plus(
        minutes,
        ChronoUnit.MINUTES
    )

    val met = Metric {
        metricName = customMetricName
        namespace = customMetricNamespace
    }

    val metStat = MetricStat {
        stat = "Maximum"
        period = 1
        metric = met
    }

    val dataQUery = MetricDataQuery {
        metricStat = metStat
        id = "foo2"
        returnData = true
    }

    val dq = ArrayList<MetricDataQuery>()
    dq.add(dataQUery)
    val getMetReq = GetMetricDataRequest {
        maxDatapoints = 10
        scanBy = ScanBy.TimestampDescending
        startTime = aws.smithy.kotlin.runtime.time.Instant(nowDate)
```

```
        endTime = aws.smithy.kotlin.runtime.time.Instant(date2)
        metricDataQueries = dq
    }

    CloudWatchClient { region = "us-east-1" }.use { cwClient ->
        val response = cwClient.getMetricData(getMetReq)
        response.metricDataResults?.forEach { item ->
            println("The label is ${item.label}")
            println("The status code is ${item.statusCode}")
        }
    }
}

suspend fun describeAlarms() {
    val typeList = ArrayList<AlarmType>()
    typeList.add(AlarmType.MetricAlarm)
    val alarmsRequest = DescribeAlarmsRequest {
        alarmTypes = typeList
        maxRecords = 10
    }

    CloudWatchClient { region = "us-east-1" }.use { cwClient ->
        val response = cwClient.describeAlarms(alarmsRequest)
        response.metricAlarms?.forEach { alarm ->
            println("Alarm name: ${alarm.alarmName}")
            println("Alarm description: ${alarm.alarmDescription}")
        }
    }
}

suspend fun createAlarm(fileName: String): String {
    // Read values from the JSON file.
    val parser = JsonFactory().createParser(File(fileName))
    val rootNode: JsonNode = ObjectMapper().readTree(parser)
    val customMetricNamespace =
 rootNode.findValue("customMetricNamespace").asText()
    val customMetricName = rootNode.findValue("customMetricName").asText()
    val alarmNameVal = rootNode.findValue("exampleAlarmName").asText()
    val emailTopic = rootNode.findValue("emailTopic").asText()
    val accountId = rootNode.findValue("accountId").asText()
    val region2 = rootNode.findValue("region").asText()

    // Create a List for alarm actions.
    val alarmActionObs: MutableList<String> = ArrayList()
```

```
    alarmActionObs.add("arn:aws:sns:$region2:$accountId:$emailTopic")
    val alarmRequest = PutMetricAlarmRequest {
        alarmActions = alarmActionObs
        alarmDescription = "Example metric alarm"
        alarmName = alarmNameVal
        comparisonOperator = ComparisonOperator.GreaterThanOrEqualToThreshold
        threshold = 100.00
        metricName = customMetricName
        namespace = customMetricNamespace
        evaluationPeriods = 1
        period = 10
        statistic = Statistic.Maximum
        datapointsToAlarm = 1
        treatMissingData = "ignore"
    }

    CloudWatchClient { region = "us-east-1" }.use { cwClient ->
        cwClient.putMetricAlarm(alarmRequest)
        println("$alarmNameVal was successfully created!")
        return alarmNameVal
    }
}

suspend fun addMetricToDashboard(fileNameVal: String, dashboardNameVal: String) {
    val dashboardRequest = PutDashboardRequest {
        dashboardName = dashboardNameVal
        dashboardBody = readFileAsString(fileNameVal)
    }

    CloudWatchClient { region = "us-east-1" }.use { cwClient ->
        cwClient.putDashboard(dashboardRequest)
        println("$dashboardNameVal was successfully updated.")
    }
}

suspend fun createNewCustomMetric(dataPoint: Double) {
    val dimension = Dimension {
        name = "UNIQUE_PAGES"
        value = "URLS"
    }

    // Set an Instant object.
    val time =
 ZonedDateTime.now(ZoneOffset.UTC).format(DateTimeFormatter.ISO_INSTANT)
```

```
    val instant = Instant.parse(time)
    val datum = MetricDatum {
        metricName = "PAGES_VISITED"
        unit = StandardUnit.None
        value = dataPoint
        timestamp = aws.smithy.kotlin.runtime.time.Instant(instant)
        dimensions = listOf(dimension)
    }

    val request = PutMetricDataRequest {
        namespace = "SITE/TRAFFIC"
        metricData = listOf(datum)
    }

    CloudWatchClient { region = "us-east-1" }.use { cwClient ->
        cwClient.putMetricData(request)
        println("Added metric values for for metric PAGES_VISITED")
    }
}

suspend fun listDashboards() {
    CloudWatchClient { region = "us-east-1" }.use { cwClient ->
        cwClient.listDashboardsPaginated({})
            .transform { it.dashboardEntries?.forEach { obj -> emit(obj) } }
            .collect { obj ->
                println("Name is ${obj.dashboardName}")
                println("Dashboard ARN is ${obj.dashboardArn}")
            }
    }
}

suspend fun createDashboardWithMetrics(dashboardNameVal: String, fileNameVal:
 String) {
    val dashboardRequest = PutDashboardRequest {
        dashboardName = dashboardNameVal
        dashboardBody = readFileAsString(fileNameVal)
    }

    CloudWatchClient { region = "us-east-1" }.use { cwClient ->
        val response = cwClient.putDashboard(dashboardRequest)
        println("$dashboardNameVal was successfully created.")
        val messages = response.dashboardValidationMessages
        if (messages != null) {
            if (messages.isEmpty()) {
```

```
                println("There are no messages in the new Dashboard")
            } else {
                for (message in messages) {
                    println("Message is: ${message.message}")
                }
            }
        }
    }
}

fun readFileAsString(file: String): String {
    return String(Files.readAllBytes(Paths.get(file)))
}

suspend fun getMetricStatistics(costDateWeek: String?) {
    val start = Instant.parse(costDateWeek)
    val endDate = Instant.now()
    val dimension = Dimension {
        name = "Currency"
        value = "USD"
    }

    val dimensionList: MutableList<Dimension> = ArrayList()
    dimensionList.add(dimension)

    val statisticsRequest = GetMetricStatisticsRequest {
        metricName = "EstimatedCharges"
        namespace = "AWS/Billing"
        dimensions = dimensionList
        statistics = listOf(Statistic.Maximum)
        startTime = aws.smithy.kotlin.runtime.time.Instant(start)
        endTime = aws.smithy.kotlin.runtime.time.Instant(endDate)
        period = 86400
    }
    CloudWatchClient { region = "us-east-1" }.use { cwClient ->
        val response = cwClient.getMetricStatistics(statisticsRequest)
        val data: List<Datapoint>? = response.datapoints
        if (data != null) {
            if (!data.isEmpty()) {
                for (datapoint in data) {
                    println("Timestamp:  ${datapoint.timestamp} Maximum value:
 ${datapoint.maximum}")
                }
            } else {
```

```
                println("The returned data list is empty")
            }
        }
    }
}

suspend fun getAndDisplayMetricStatistics(nameSpaceVal: String, metVal: String,
 metricOption: String, date: String, myDimension: Dimension) {
    val start = Instant.parse(date)
    val endDate = Instant.now()
    val statisticsRequest = GetMetricStatisticsRequest {
        endTime = aws.smithy.kotlin.runtime.time.Instant(endDate)
        startTime = aws.smithy.kotlin.runtime.time.Instant(start)
        dimensions = listOf(myDimension)
        metricName = metVal
        namespace = nameSpaceVal
        period = 86400
        statistics = listOf(Statistic.fromValue(metricOption))
    }

    CloudWatchClient { region = "us-east-1" }.use { cwClient ->
        val response = cwClient.getMetricStatistics(statisticsRequest)
        val data = response.datapoints
        if (data != null) {
            if (data.isNotEmpty()) {
                for (datapoint in data) {
                    println("Timestamp: ${datapoint.timestamp} Maximum value:
 ${datapoint.maximum}")
                }
            } else {
                println("The returned data list is empty")
            }
        }
    }
}

suspend fun listMets(namespaceVal: String?): ArrayList<String>? {
    val metList = ArrayList<String>()
    val request = ListMetricsRequest {
        namespace = namespaceVal
    }
    CloudWatchClient { region = "us-east-1" }.use { cwClient ->
        val reponse = cwClient.listMetrics(request)
        reponse.metrics?.forEach { metrics ->
```

```
            val data = metrics.metricName
            if (!metList.contains(data)) {
                metList.add(data!!)
            }
        }
    }
    return metList
}

suspend fun getSpecificMet(namespaceVal: String?): Dimension? {
    val request = ListMetricsRequest {
        namespace = namespaceVal
    }
    CloudWatchClient { region = "us-east-1" }.use { cwClient ->
        val response = cwClient.listMetrics(request)
        val myList = response.metrics
        if (myList != null) {
            return myList[0].dimensions?.get(0)
        }
    }
    return null
}

suspend fun listNameSpaces(): ArrayList<String> {
    val nameSpaceList = ArrayList<String>()
    CloudWatchClient { region = "us-east-1" }.use { cwClient ->
        val response = cwClient.listMetrics(ListMetricsRequest {})
        response.metrics?.forEach { metrics ->
            val data = metrics.namespace
            if (!nameSpaceList.contains(data)) {
                nameSpaceList.add(data!!)
            }
        }
    }
    return nameSpaceList
}
```

- API の詳細については、「AWS SDK for Kotlin API リファレンス」の以下のトピックを参照してください。
  - DeleteAlarms
  - DeleteAnomalyDetector

- DeleteDashboards
- DescribeAlarmHistory
- DescribeAlarms
- DescribeAlarmsForMetric
- DescribeAnomalyDetectors
- GetMetricData
- GetMetricStatistics
- GetMetricWidgetImage
- ListMetrics
- PutAnomalyDetector
- PutDashboard
- PutMetricAlarm
- PutMetricData

AWS SDK デベロッパーガイドとコード例の完全なリストについては、「AWS SDK で CloudWatch を使用する」を参照してください。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK により CloudWatch メトリクスとアラームを管理する

次のコードサンプルは、以下の操作方法を示しています。

- アラームを作成して CloudWatch メトリクスを監視します。
- メトリクスにデータを配置し、アラームをトリガーします。
- アラームからデータを取得します。
- アラームを削除します。

Python

SDK for Python (Boto3)

> **Note**
>
> GitHub には、その他のリソースもあります。完全な例を見つけて、AWS コード例リポジトリでの設定と実行の方法を確認してください。

CloudWatch オペレーションをラップするクラスを作成します。

```
from datetime import datetime, timedelta
import logging
from pprint import pprint
import random
import time
import boto3
from botocore.exceptions import ClientError

logger = logging.getLogger(__name__)


class CloudWatchWrapper:
    """Encapsulates Amazon CloudWatch functions."""

    def __init__(self, cloudwatch_resource):
        """
        :param cloudwatch_resource: A Boto3 CloudWatch resource.
        """
        self.cloudwatch_resource = cloudwatch_resource


    def put_metric_data_set(self, namespace, name, timestamp, unit, data_set):
        """
        Sends a set of data to CloudWatch for a metric. All of the data in the
 set
        have the same timestamp and unit.

        :param namespace: The namespace of the metric.
        :param name: The name of the metric.
        :param timestamp: The UTC timestamp for the metric.
        :param unit: The unit of the metric.
```

```
        :param data_set: The set of data to send. This set is a dictionary that
                         contains a list of values and a list of corresponding
counts.
                         The value and count lists must be the same length.
        """
        try:
            metric = self.cloudwatch_resource.Metric(namespace, name)
            metric.put_data(
                Namespace=namespace,
                MetricData=[
                    {
                        "MetricName": name,
                        "Timestamp": timestamp,
                        "Values": data_set["values"],
                        "Counts": data_set["counts"],
                        "Unit": unit,
                    }
                ],
            )
            logger.info("Put data set for metric %s.%s.", namespace, name)
        except ClientError:
            logger.exception("Couldn't put data set for metric %s.%s.",
namespace, name)
            raise


    def create_metric_alarm(
        self,
        metric_namespace,
        metric_name,
        alarm_name,
        stat_type,
        period,
        eval_periods,
        threshold,
        comparison_op,
    ):
        """
        Creates an alarm that watches a metric.

        :param metric_namespace: The namespace of the metric.
        :param metric_name: The name of the metric.
        :param alarm_name: The name of the alarm.
        :param stat_type: The type of statistic the alarm watches.
```

```
        :param period: The period in which metric data are grouped to calculate
                       statistics.
        :param eval_periods: The number of periods that the metric must be over
the
                             alarm threshold before the alarm is set into an
alarmed
                             state.
        :param threshold: The threshold value to compare against the metric
statistic.
        :param comparison_op: The comparison operation used to compare the
threshold
                              against the metric.
        :return: The newly created alarm.
        """
        try:
            metric = self.cloudwatch_resource.Metric(metric_namespace,
metric_name)
            alarm = metric.put_alarm(
                AlarmName=alarm_name,
                Statistic=stat_type,
                Period=period,
                EvaluationPeriods=eval_periods,
                Threshold=threshold,
                ComparisonOperator=comparison_op,
            )
            logger.info(
                "Added alarm %s to track metric %s.%s.",
                alarm_name,
                metric_namespace,
                metric_name,
            )
        except ClientError:
            logger.exception(
                "Couldn't add alarm %s to metric %s.%s",
                alarm_name,
                metric_namespace,
                metric_name,
            )
            raise
        else:
            return alarm


    def put_metric_data(self, namespace, name, value, unit):
```

```
        """
        Sends a single data value to CloudWatch for a metric. This metric is
given
        a timestamp of the current UTC time.

        :param namespace: The namespace of the metric.
        :param name: The name of the metric.
        :param value: The value of the metric.
        :param unit: The unit of the metric.
        """
        try:
            metric = self.cloudwatch_resource.Metric(namespace, name)
            metric.put_data(
                Namespace=namespace,
                MetricData=[{"MetricName": name, "Value": value, "Unit": unit}],
            )
            logger.info("Put data for metric %s.%s", namespace, name)
        except ClientError:
            logger.exception("Couldn't put data for metric %s.%s", namespace,
name)
            raise


    def get_metric_statistics(self, namespace, name, start, end, period,
stat_types):
        """
        Gets statistics for a metric within a specified time span. Metrics are
grouped
        into the specified period.

        :param namespace: The namespace of the metric.
        :param name: The name of the metric.
        :param start: The UTC start time of the time span to retrieve.
        :param end: The UTC end time of the time span to retrieve.
        :param period: The period, in seconds, in which to group metrics. The
period
                       must match the granularity of the metric, which depends on
                       the metric's age. For example, metrics that are older than
                       three hours have a one-minute granularity, so the period
must
                       be at least 60 and must be a multiple of 60.
        :param stat_types: The type of statistics to retrieve, such as average
value
                           or maximum value.
```

```
        :return: The retrieved statistics for the metric.
        """
        try:
            metric = self.cloudwatch_resource.Metric(namespace, name)
            stats = metric.get_statistics(
                StartTime=start, EndTime=end, Period=period,
Statistics=stat_types
            )
            logger.info(
                "Got %s statistics for %s.", len(stats["Datapoints"]),
stats["Label"]
            )
        except ClientError:
            logger.exception("Couldn't get statistics for %s.%s.", namespace,
name)
            raise
        else:
            return stats


    def get_metric_alarms(self, metric_namespace, metric_name):
        """
        Gets the alarms that are currently watching the specified metric.

        :param metric_namespace: The namespace of the metric.
        :param metric_name: The name of the metric.
        :returns: An iterator that yields the alarms.
        """
        metric = self.cloudwatch_resource.Metric(metric_namespace, metric_name)
        alarm_iter = metric.alarms.all()
        logger.info("Got alarms for metric %s.%s.", metric_namespace,
metric_name)
        return alarm_iter


    def delete_metric_alarms(self, metric_namespace, metric_name):
        """
        Deletes all of the alarms that are currently watching the specified
metric.

        :param metric_namespace: The namespace of the metric.
        :param metric_name: The name of the metric.
        """
        try:
```

```
            metric = self.cloudwatch_resource.Metric(metric_namespace,
 metric_name)
            metric.alarms.delete()
            logger.info(
                "Deleted alarms for metric %s.%s.", metric_namespace, metric_name
            )
        except ClientError:
            logger.exception(
                "Couldn't delete alarms for metric %s.%s.",
                metric_namespace,
                metric_name,
            )
            raise
```

このラッパークラスを使用して、メトリクスにデータを配置し、メトリクスを監視するアラームをトリガーし、そのアラームからデータを取得します。

```
def usage_demo():
    print("-" * 88)
    print("Welcome to the Amazon CloudWatch metrics and alarms demo!")
    print("-" * 88)

    logging.basicConfig(level=logging.INFO, format="%(levelname)s: %(message)s")

    cw_wrapper = CloudWatchWrapper(boto3.resource("cloudwatch"))

    minutes = 20
    metric_namespace = "doc-example-metric"
    metric_name = "page_views"
    start = datetime.utcnow() - timedelta(minutes=minutes)
    print(
        f"Putting data into metric {metric_namespace}.{metric_name} spanning the
 "
        f"last {minutes} minutes."
    )
    for offset in range(0, minutes):
        stamp = start + timedelta(minutes=offset)
        cw_wrapper.put_metric_data_set(
            metric_namespace,
            metric_name,
```

```
            stamp,
            "Count",
            {
                "values": [
                    random.randint(bound, bound * 2)
                    for bound in range(offset + 1, offset + 11)
                ],
                "counts": [random.randint(1, offset + 1) for _ in range(10)],
            },
        )

    alarm_name = "high_page_views"
    period = 60
    eval_periods = 2
    print(f"Creating alarm {alarm_name} for metric {metric_name}.")
    alarm = cw_wrapper.create_metric_alarm(
        metric_namespace,
        metric_name,
        alarm_name,
        "Maximum",
        period,
        eval_periods,
        100,
        "GreaterThanThreshold",
    )
    print(f"Alarm ARN is {alarm.alarm_arn}.")
    print(f"Current alarm state is: {alarm.state_value}.")

    print(
        f"Sending data to trigger the alarm. This requires data over the
threshold "
        f"for {eval_periods} periods of {period} seconds each."
    )
    while alarm.state_value == "INSUFFICIENT_DATA":
        print("Sending data for the metric.")
        cw_wrapper.put_metric_data(
            metric_namespace, metric_name, random.randint(100, 200), "Count"
        )
        alarm.load()
        print(f"Current alarm state is: {alarm.state_value}.")
        if alarm.state_value == "INSUFFICIENT_DATA":
            print(f"Waiting for {period} seconds...")
            time.sleep(period)
        else:
```

```
            print("Wait for a minute for eventual consistency of metric data.")
            time.sleep(period)
            if alarm.state_value == "OK":
                alarm.load()
                print(f"Current alarm state is: {alarm.state_value}.")

    print(
        f"Getting data for metric {metric_namespace}.{metric_name} during
timespan "
        f"of {start} to {datetime.utcnow()} (times are UTC)."
    )
    stats = cw_wrapper.get_metric_statistics(
        metric_namespace,
        metric_name,
        start,
        datetime.utcnow(),
        60,
        ["Average", "Minimum", "Maximum"],
    )
    print(
        f"Got {len(stats['Datapoints'])} data points for metric "
        f"{metric_namespace}.{metric_name}."
    )
    pprint(sorted(stats["Datapoints"], key=lambda x: x["Timestamp"]))

    print(f"Getting alarms for metric {metric_name}.")
    alarms = cw_wrapper.get_metric_alarms(metric_namespace, metric_name)
    for alarm in alarms:
        print(f"Alarm {alarm.name} is currently in state {alarm.state_value}.")

    print(f"Deleting alarms for metric {metric_name}.")
    cw_wrapper.delete_metric_alarms(metric_namespace, metric_name)

    print("Thanks for watching!")
    print("-" * 88)
```

- API の詳細については、「AWS SDK for Python (Boto3) API リファレンス」の以下のトピックを参照してください。
  - DeleteAlarms

- DescribeAlarmsForMetric
- DisableAlarmActions
- EnableAlarmActions
- GetMetricStatistics
- ListMetrics
- PutMetricAlarm
- PutMetricData

AWS SDK デベロッパーガイドとコード例の完全なリストについては、「AWS SDK で CloudWatch を使用する」を参照してください。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK を使用した CloudWatch のクロスサービスの例

次のサンプルアプリケーションでは、AWS SDK を使用して CloudWatch を他の AWS のサービス と組み合わせます。それぞれの例には、GitHub へのリンクがあり、アプリケーションを設定および実行する方法についての説明を参照できます。

例

- AWS SDK を使用した Amazon DynamoDB のパフォーマンスのモニタリング

AWS SDK を使用した Amazon DynamoDB のパフォーマンスのモニタリング

次のコード例は、DynamoDB のパフォーマンスをモニタリングするように、アプリケーションを設定する方法を示しています。

Java

SDK for Java 2.x

この例では、DynamoDB のパフォーマンスをモニタリングするように Java アプリケーションを設定する方法を示します。アプリケーションからメトリクスデータを CloudWatch に送信してパフォーマンスをモニタリングできます。

完全なソースコードとセットアップおよび実行の手順については、GitHub で完全な例を参照してください。

この例で使用されているサービス

- CloudWatch
- DynamoDB

AWS SDK デベロッパーガイドとコード例の完全なリストについては、「AWS SDK で CloudWatch を使用する」を参照してください。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

Amazon CloudWatch のセキュリティ

AWS では、クラウドのセキュリティが最優先事項です。セキュリティを最も重視する組織の要件を満たすために構築された AWS のデータセンターとネットワークアーキテクチャは、お客様に大きく貢献します。

セキュリティは、AWS とお客様とが共有する責務です。責任共有モデルでは、これをクラウドのセキュリティおよびクラウド内のセキュリティと説明しています。

- クラウドのセキュリティ - AWS は、AWS クラウドで AWS サービスを実行するインフラストラクチャを保護する責任を負います。また AWS は、安全に使用できるサービスを提供します。AWS コンプライアンスプログラムの一環として、サードパーティー監査者が定期的にセキュリティの有効性をテストおよび検証します。CloudWatch に適用されるコンプライアンスプログラムの詳細については、「コンプライアンスプログラムによる対象範囲内の AWS のサービス」「」を参照してください。
- クラウド内のセキュリティ – お客様の責任範囲は、ご使用の AWS のサービスに応じて異なります。また、お客様は、お客様のデータの機密性、企業の要件、および適用可能な法律および規制などの他の要因についても責任を担います。

このドキュメントは、Amazon CloudWatch を使用する際の責任共有モデルの適用方法を理解するのに役立ちます。ここでは、セキュリティとコンプライアンスの目標を満たすように Amazon CloudWatch を設定する方法について説明します。また、CloudWatch リソースのモニタリングや保護に他の AWS サービスを利用する方法についても説明します。

目次

Amazon CloudWatch におけるデータ保護

AWS 責任共有モデルは、Amazon CloudWatch のデータ保護に適用されます。このモデルで説明されているように、AWS は、AWS クラウド のすべてを実行するグローバルインフラストラクチャを保護するがあります。お客様は、このインフラストラクチャでホストされているコンテンツに対する管理を維持する責任があります。また、使用する AWS のサービスのセキュリティ設定と管理タスクもユーザーの責任となります。データプライバシーの詳細については、「データプライバシーのよくある質問」を参照してください。欧州でのデータ保護の詳細については、「AWS セキュリティブログ」に投稿された「AWS 責任共有モデルおよび GDPR」のブログ記事を参照してください。

データを保護するため、AWS アカウント の認証情報を保護し、AWS IAM Identity Center または AWS Identity and Access Management (IAM) を使用して個々のユーザーをセットアップすることをお勧めします。この方法により、それぞれのジョブを遂行するために必要な権限のみを各ユーザーに付与できます。また、次の方法でデータを保護することをおすすめします。

- 各アカウントで多要素認証 (MFA) を使用します。
- SSL/TLS を使用して AWS リソースと通信します。TLS 1.2 が必須です。TLS 1.3 が推奨されます。
- AWS CloudTrail で API とユーザーアクティビティロギングをセットアップします。
- AWS のサービス内でデフォルトである、すべてのセキュリティ管理に加え、AWS の暗号化ソリューションを使用します。
- Amazon Macie などの高度なマネージドセキュリティサービスを使用します。これらは、Amazon S3 に保存されている機密データの検出と保護を支援します。
- コマンドラインインターフェイスまたは API により AWS にアクセスするときに FIPS 140-2 検証済み暗号化モジュールが必要な場合は、FIPS エンドポイントを使用します。利用可能な FIPS エンドポイントの詳細については、「連邦情報処理規格 (FIPS) 140-2」を参照してください。

顧客の E メールアドレスなどの機密情報や重要情報は、タグや Name フィールドなどの自由形式のフィールドに入力しないことを強くお勧めします。これには、コンソール、API、AWS CLI、または AWS SDK を使用して、CloudWatch またはその他の AWS のサービスを使用する場合も含まれます。名前に使用する自由記述のテキストフィールドやタグに入力したデータは、課金や診断ログに使用される場合があります。外部サーバーへ URL を提供する場合は、そのサーバーへのリクエストを有効にするために認証情報を URL に含めないことを強くお勧めします。

転送中の暗号化

は、転送中のデータのエンドツーエンドの暗号化を使用します。

Amazon CloudWatch 用 Identity and Access Management

AWS Identity and Access Management (IAM) は、管理者が AWS リソースへのアクセスを安全に制御するために役立つ AWS のサービスです。IAM 管理者は、誰を認証 (サインインを許可) し、誰に CloudWatch リソースの使用を許可する (アクセス許可を付与する) かを制御します。IAM は、追加費用なしで使用できる AWS のサービスです。

トピック

対象者

AWS Identity and Access Management (IAM) の使用方法は、CloudWatch で行う作業によって異なります。

サービスユーザー – ジョブを実行するために CloudWatch サービスを使用する場合は、管理者から必要な認証情報とアクセス許可が与えられます。作業を実行するためにさらに多くの CloudWatch 機能を使用する場合、追加のアクセス許可が必要になることがあります。アクセスの管理方法を理解すると、管理者から適切な権限をリクエストするのに役に立ちます。CloudWatch の機能にアクセスできない場合は、「Amazon CloudWatch のアイデンティティとアクセスのトラブルシューティング」を参照してください。

サービス管理者 - 社内の CloudWatch リソースを担当している管理者は、通常、CloudWatch にフルアクセスできます。サービスユーザーが CloudWatch のどの機能やリソースにアクセスするかを決めるのは管理者の仕事です。その後、IAM 管理者にリクエストを送信して、サービスユーザーの権限を変更する必要があります。このページの情報を点検して、IAM の基本概念を理解してください。CloudWatch で IAM をどのように自社で利用できるかの詳細については、「Amazon CloudWatch と IAM の連携方法」を参照してください。

IAM 管理者 - IAM 管理者は、CloudWatch へのアクセスを管理するポリシーの詳しい作成方法について確認する場合があります。IAM で使用できる CloudWatch アイデンティティベースのポリシー例については、「Amazon CloudWatch のアイデンティティベースのポリシー例」を参照してください。

アイデンティティによる認証

認証とは、アイデンティティ認証情報を使用して AWSにサインインする方法です。ユーザーは、AWS アカウントのルートユーザー として、IAM ユーザーとして、または IAM ロールを引き受けることによって、認証済み (AWS にサインイン済み) である必要があります。

ID ソースから提供された認証情報を使用すると、フェデレーティッドアイデンティティとして AWS にサインインできます。AWS IAM Identity Centerフェデレーティッドアイデンティティの例としては、(IAM アイデンティティセンター) ユーザー、貴社のシングルサインオン認証、Google または Facebook の認証情報などがあります。フェデレーティッドアイデンティティとしてサインインする場合、IAM ロールを使用して、前もって管理者により ID フェデレーションが設定されています。フェデレーションを使用して AWSにアクセスする場合、間接的にロールを引き受けることになります。

ユーザーのタイプに応じて、AWS Management Console または AWSアクセスポータルにサインインできます。AWS へのサインインの詳細については、『AWS サインイン ユーザーガイド』の「AWS アカウント にサインインする方法」を参照してください。

プログラムで AWS にアクセスする場合、AWS は Software Development Kit (SDK) とコマンドラインインターフェイス (CLI) を提供し、認証情報でリクエストに暗号で署名します。AWS ツールを使

用しない場合は、リクエストに自分で署名する必要があります。リクエストに署名する推奨方法の使用については、『IAM ユーザーガイド』の「AWS API リクエストの署名」を参照してください。

使用する認証方法を問わず、追加セキュリティ情報の提供をリクエストされる場合もあります。例えば、AWS では、アカウントのセキュリティ強化のために多要素認証 (MFA) の使用をお勧めしています。詳細については、『AWS IAM Identity Center ユーザーガイド』の「Multi-factor authentication」(多要素認証) および『IAM ユーザーガイド』の「AWS における多要素認証 (MFA) の使用」を参照してください。

AWS アカウントのルートユーザー

AWS アカウントを作成する場合は、このアカウントのすべての AWS のサービスとリソースに対して完全なアクセス権を持つ 1 つのサインインアイデンティティから始めます。この ID は AWS アカウント ルートユーザーと呼ばれ、アカウントの作成に使用したメールアドレスとパスワードでサインインすることによってアクセスできます。日常的なタスクには、ルートユーザーを使用しないことを強くお勧めします。ルートユーザーの認証情報は保護し、ルートユーザーでしか実行できないタスクを実行するときに使用します。ルートユーザーとしてサインインする必要があるタスクの完全なリストについては、『IAM ユーザーガイド』の「ルートユーザー認証情報が必要なタスク」を参照してください。

フェデレーティッドアイデンティティ

ベストプラクティスとして、管理者アクセスを必要とするユーザーを含む人間のユーザーに対し、ID プロバイダーとのフェデレーションを使用して、一時認証情報の使用により、AWS のサービス にアクセスすることを要求します。

フェデレーテッドアイデンティティ は、エンタープライズユーザーディレクトリ、ウェブ ID プロバイダー、AWS Directory Service、アイデンティティセンターディレクトリのユーザーか、または ID ソースから提供された認証情報を使用して AWS のサービス にアクセスするユーザーです。フェデレーテッドアイデンティティ が AWS アカウント にアクセスすると、ロールが継承され、ロールは一時認証情報を提供します。

アクセスを一元管理する場合は、AWS IAM Identity Center を使用することをお勧めします。IAM アイデンティティセンター でユーザーとグループを作成するか、すべての AWS アカウント とアプリケーションで使用するために、独自の ID ソースで一連のユーザーとグループに接続して同期することもできます。IAM Identity Center の詳細については、『AWS IAM Identity Center ユーザーガイド』の「What is IAM Identity Center?」(IAM Identity Center とは) を参照してください。

IAM ユーザーとグループ

IAM ユーザーは、1 人のユーザーまたは 1 つのアプリケーションに対して特定の権限を持つ AWS アカウント 内のアイデンティティです。可能であれば、パスワードやアクセスキーなどの長期的な認証情報を保有する IAM ユーザーを作成する代わりに、一時認証情報を使用することをお勧めします。ただし、IAM ユーザーでの長期的な認証情報が必要な特定のユースケースがある場合は、アクセスキーをローテーションすることをお勧めします。詳細については、「IAM ユーザーガイド」の「長期的な認証情報を必要とするユースケースのためにアクセスキーを定期的にローテーションする」を参照してください。

IAM グループは、IAM ユーザーの集団を指定するアイデンティティです。グループとしてサインインすることはできません。グループを使用して、複数のユーザーに対して一度に権限を指定できます。多数のユーザーグループがある場合、グループを使用することで権限の管理が容易になります。例えば、IAMAdmins という名前のグループを設定して、そのグループに IAM リソースを管理する権限を与えることができます。

ユーザーは、ロールとは異なります。ユーザーは 1 人の人または 1 つのアプリケーションに一意に関連付けられますが、ロールはそれを必要とする任意の人が引き受けるようになっています。ユーザーには永続的な長期の認証情報がありますが、ロールでは一時的な認証情報が提供されます。詳細については、『IAM ユーザーガイド』の「IAM ユーザー (ロールではなく) の作成が適している場合」を参照してください。

IAM ロール

IAM ロールは、特定の権限を持つ、AWS アカウント 内のアイデンティティです。これは IAM ユーザーに似ていますが、特定のユーザーには関連付けられていません。ロールの切り替えによって、AWS Management Console で IAM ロールを一時的に引き受けることができます。ロールを引き受けるには、AWS CLI または AWSAPI オペレーションを呼び出すか、カスタム URL を使用します。ロールを使用する方法の詳細については、「IAM ユーザーガイド」の「IAM ロールの使用」を参照してください。

IAM ロールと一時的な認証情報は、次の状況で役立ちます:

- フェデレーションユーザーアクセス – フェデレーティッドアイデンティティに権限を割り当てるには、ロールを作成してそのロールの権限を定義します。フェデレーティッドアイデンティティが認証されると、そのアイデンティティはロールに関連付けられ、ロールで定義されている権限が付与されます。フェデレーションの詳細については、『IAM ユーザーガイド』の「サードパーティーアイデンティティプロバイダー向けロールの作成」 を参照してください。IAM アイデンティティセンターを使用する場合、権限セットを設定します。アイデンティティが認証後にアク

セスできるものを制御するため、IAM Identity Center は、権限セットを IAM のロールに関連付けます。権限セットの詳細については、『AWS IAM Identity Center ユーザーガイド』の「権限セット」を参照してください。

- 一時的な IAM ユーザー権限 - IAM ユーザーまたはロールは、特定のタスクに対して複数の異なる権限を一時的に IAM ロールで引き受けることができます。
- クロスアカウントアクセス - IAM ロールを使用して、自分のアカウントのリソースにアクセスすることを、別のアカウントの人物 (信頼済みプリンシパル) に許可できます。クロスアカウントアクセス権を付与する主な方法は、ロールを使用することです。ただし、一部の AWS のサービスでは、(ロールをプロキシとして使用する代わりに) リソースにポリシーを直接アタッチできます。クロスアカウントアクセスにおけるロールとリソースベースのポリシーの違いについては、『IAM ユーザーガイド』の「IAM ロールとリソースベースのポリシーとの相違点」を参照してください。
- クロスサービスアクセス権 - 一部の AWS のサービスでは、他の AWS のサービスの機能を使用します。例えば、あるサービスで呼び出しを行うと、通常そのサービスによって Amazon EC2 でアプリケーションが実行されたり、Amazon S3 にオブジェクトが保存されたりします。サービスでは、呼び出し元プリンシパルの権限、サービスロール、またはサービスにリンクされたロールを使用してこれを行う場合があります。
  - 転送アクセスセッション (FAS) – IAM ユーザーまたはロールを使用して AWS でアクションを実行するユーザーは、プリンシパルと見なされます。一部のサービスを使用する際に、アクションを実行することで、別のサービスの別のアクションがトリガーされることがあります。FAS は、AWS のサービスを呼び出すプリンシパルの権限を、AWS のサービスのリクエストと合わせて使用し、ダウンストリームのサービスに対してリクエストを行います。FAS リクエストは、サービスが、完了するために他の AWS のサービス またはリソースとのやりとりを必要とするリクエストを受け取ったときにのみ行われます。この場合、両方のアクションを実行するためのアクセス許可が必要です。FAS リクエストを行う際のポリシーの詳細については、「転送アクセスセッション」を参照してください。
  - サービスロール - サービスがユーザーに代わってアクションを実行するために引き受ける IAM ロールです。IAM 管理者は、IAM 内からサービスロールを作成、変更、削除できます。詳細については、「IAM ユーザーガイド」の「AWS のサービス に権限を委任するロールの作成」を参照してください。
  - サービスリンクロール - サービスリンクロールは、AWS のサービス にリンクされたサービスロールの一種です。サービスがロールを引き受け、ユーザーに代わってアクションを実行できるようになります。サービスリンクロールは、AWS アカウント に表示され、サービスによって所有されます。IAM 管理者は、サービスにリンクされたロールの権限を表示できますが、編集することはできません。

- Amazon EC2 で実行されるアプリケーション - EC2 インスタンスで実行され、AWS CLI または AWS API 要求を行っているアプリケーションの一時的な認証情報を管理するために、IAM ロールを使用できます。これは、EC2 インスタンス内でのアクセスキーの保存に推奨されます。AWS ロールを EC2 インスタンスに割り当て、そのすべてのアプリケーションで使用できるようにするには、インスタンスに添付されたインスタンスプロファイルを作成します。インスタンスプロファイルにはロールが含まれ、EC2 インスタンスで実行されるプログラムは一時的な認証情報を取得できます。詳細については、『IAM ユーザーガイド』の「Amazon EC2 インスタンスで実行されるアプリケーションに IAM ロールを使用して権限を付与する」を参照してください。

IAM ロールと IAM ユーザーのどちらを使用するかについては、『IAM ユーザーガイド』の「(IAM ユーザーではなく) IAM ロールをいつ作成したら良いのか?」を参照してください。

ポリシーを使用したアクセス権の管理

AWS でアクセス権を管理するには、ポリシーを作成して AWS アイデンティティまたはリソースにアタッチします。ポリシーは AWS のオブジェクトであり、アイデンティティやリソースに関連付けて、これらの権限を定義します。AWS は、プリンシパル (ユーザー、ルートユーザー、またはロールセッション) がリクエストを行うと、これらのポリシーを評価します。ポリシーでの権限により、リクエストが許可されるか拒否されるかが決まります。大半のポリシーは JSON ドキュメントとして AWSに保存されます。JSON ポリシードキュメントの構造と内容の詳細については、『IAM ユーザーガイド』の「JSON ポリシー概要」を参照してください。

管理者は AWSJSON ポリシーを使用して、だれが何にアクセスできるかを指定できます。つまり、どのプリンシパルがどんなリソースにどんな条件でアクションを実行できるかということです。

デフォルトでは、ユーザーやロールに権限はありません。IAM 管理者は、リソースで必要なアクションを実行するための権限をユーザーに付与する IAM ポリシーを作成できます。その後、管理者はロールに IAM ポリシーを追加し、ユーザーはロールを引き継ぐことができます。

IAM ポリシーは、オペレーションの実行方法を問わず、アクションの権限を定義します。例えば、`iam:GetRole` アクションを許可するポリシーがあるとします。このポリシーがあるユーザーは、AWS Management Console、AWS CLI、または AWS API からロール情報を取得できます。

アイデンティティベースポリシー

アイデンティティベースポリシーは、IAM ユーザー、ユーザーのグループ、ロールなど、アイデンティティにアタッチできる JSON 権限ポリシードキュメントです。これらのポリシーは、ユーザーとロールが実行できるアクション、リソース、および条件をコントロールします。アイデンティティ

ベースのポリシーを作成する方法については、「IAM ユーザーガイド」の「IAM ポリシーの作成」を参照してください。

アイデンティティベースポリシーは、さらにインラインポリシーまたはマネージドポリシーに分類できます。インラインポリシーは、単一のユーザー、グループ、またはロールに直接埋め込まれます。管理ポリシーは、AWS アカウント 内の複数のユーザー、グループ、およびロールにアタッチできるスタンドアロンポリシーです。マネージドポリシーには、AWS マネージドポリシーおよびカスタマーマネージドポリシーがあります。マネージドポリシーまたはインラインポリシーのいずれかを選択する方法については、『IAM ユーザーガイド』の「マネージドポリシーとインラインポリシーの比較」を参照してください。

リソースベースのポリシー

リソースベースのポリシーは、リソースに添付する JSON ポリシードキュメントです。リソースベースのポリシーには例として、IAM ロールの信頼ポリシーや Amazon S3 バケットポリシーがあげられます。リソースベースのポリシーをサポートするサービスでは、サービス管理者はポリシーを使用して特定のリソースへのアクセスを制御できます。ポリシーがアタッチされているリソースの場合、指定されたプリンシパルがそのリソースに対して実行できるアクションと条件は、ポリシーによって定義されます。リソースベースのポリシーでは、プリンシパルを指定する必要があります。プリンシパルには、アカウント、ユーザー、ロール、フェデレーションユーザー、または AWS のサービス を含めることができます。

リソースベースのポリシーは、そのサービス内にあるインラインポリシーです。リソースベースのポリシーでは IAM の AWSマネージドポリシーは使用できません。

アクセスコントロールリスト (ACL)

アクセスコントロールリスト (ACL) は、どのプリンシパル (アカウントメンバー、ユーザー、またはロール) がリソースにアクセスするための権限を持つかをコントロールします。ACL はリソースベースのポリシーに似ていますが、JSON ポリシードキュメント形式は使用しません。

Amazon S3、AWS WAF、および Amazon VPC は、ACL をサポートするサービスの例です。ACL の詳細については、『Amazon Simple Storage Service デベロッパーガイド』の「アクセスコントロールリスト (ACL) の概要」を参照してください。

その他のポリシータイプ

AWS では、他の一般的ではないポリシータイプをサポートしています。これらのポリシータイプでは、より一般的なポリシータイプで付与された最大の権限を設定できます。

- アクセス許可の境界 - アクセス許可の境界は、アイデンティティベースのポリシーによって IAM エンティティ (IAM ユーザーまたはロール) に付与できる権限の上限を設定する高度な機能です。エンティティにアクセス許可の境界を設定できます。結果として得られる権限は、エンティティのアイデンティティベースポリシーとそのアクセス許可の境界の共通部分になります。`Principal` フィールドでユーザーまたはロールを指定するリソースベースのポリシーでは、アクセス許可の境界は制限されません。これらのポリシーのいずれかを明示的に拒否した場合、権限は無効になります。アクセス許可の境界界の詳細については、『IAM ユーザーガイド』の「IAM エンティティのアクセス許可の境界」を参照してください。
- サービスコントロールポリシー (SCP) - SCP は、AWS Organizations で組織や組織単位 (OU) の最大権限を指定する JSON ポリシーです。AWS Organizations は、顧客のビジネスが所有する複数の AWS アカウント をグループ化し、一元的に管理するサービスです。組織内のすべての機能を有効にすると、サービスコントロールポリシー (SCP) を一部またはすべてのアカウントに適用できます。SCP はメンバーアカウントのエンティティに対する権限を制限します (各 AWS アカウントのルートユーザー など)。Organizations と SCP の詳細については、『AWS Organizations ユーザーガイド』の「SCP の仕組み」を参照してください。
- セッションポリシー - セッションポリシーは、ロールまたはフェデレーションユーザーの一時的なセッションをプログラムで作成する際にパラメータとして渡す高度なポリシーです。結果としてセッションの権限は、ユーザーまたはロールのアイデンティティベースポリシーとセッションポリシーの共通部分になります。また、リソースベースのポリシーから権限が派生する場合もあります。これらのポリシーのいずれかを明示的に拒否した場合、権限は無効になります。詳細については、「IAM ユーザーガイド」の「セッションポリシー」を参照してください。

複数のポリシータイプ

1 つのリクエストに複数のタイプのポリシーが適用されると、結果として作成される権限を理解するのがさらに難しくなります。複数のポリシータイプが関連するとき、リクエストを許可するかどうかを AWS が決定する方法の詳細については、「IAM ユーザーガイド」の「ポリシーの評価論理」を参照してください。

Amazon CloudWatch と IAM の連携方法

IAM を使用して CloudWatch へのアクセスを管理する前に、CloudWatch で利用できる IAM の機能を確認してください。

Amazon CloudWatch で利用できる IAM の機能

IAM の機能	CloudWatch サポート
アイデンティティベースのポリシー	Yes
リソースベースのポリシー	No
ポリシーアクション	Yes
ポリシーリソース	はい
ポリシー条件キー (サービス固有)	はい
ACL	No
ABAC (ポリシー内のタグ)	部分的
一時的な認証情報	Yes
プリンシパル権限	Yes
サービスロール	あり
サービスリンクロール	いいえ

大部分の IAM 機能が CloudWatch やその他の AWS サービスとどのように連携するかの概要については、IAM ユーザーガイドの「IAM と連携する AWS のサービス」を参照してください。

CloudWatch のアイデンティティベースのポリシー

アイデンティティベースポリシーをサポートする	Yes

アイデンティティベースポリシーは、IAM ユーザー、ユーザーグループ、ロールなど、アイデンティティにアタッチできる JSON 権限ポリシードキュメントです。これらのポリシーは、ユーザーとロールが実行できるアクション、リソース、および条件をコントロールします。アイデンティティベースのポリシーを作成する方法については、『IAM ユーザーガイド』の「IAM ポリシーの作成」を参照してください。

IAM アイデンティティベースのポリシーでは、許可または拒否するアクションとリソース、およびアクションを許可または拒否する条件を指定できます。プリンシパルは、それが添付されているユーザーまたはロールに適用されるため、アイデンティティベースのポリシーでは指定できません。JSON ポリシーで使用できるすべての要素については、「IAM ユーザーガイド」の「IAM JSON ポリシーの要素のリファレンス」を参照してください。

CloudWatch のアイデンティティベースのポリシー例

CloudWatch のアイデンティティベースのポリシー例については、「Amazon CloudWatch のアイデンティティベースのポリシー例」を参照してください。

CloudWatch 内のリソースベースのポリシー

リソースベースのポリシーのサポート	No

リソースベースのポリシーは、リソースに添付する JSON ポリシードキュメントです。リソースベースのポリシーには例として、IAM ロールの信頼ポリシーや Amazon S3 バケットポリシーがあげられます。リソースベースのポリシーをサポートするサービスでは、サービス管理者はポリシーを使用して特定のリソースへのアクセスを制御できます。ポリシーがアタッチされているリソースの場合、指定されたプリンシパルがそのリソースに対して実行できるアクションと条件は、ポリシーによって定義されます。リソースベースのポリシーでは、プリンシパルを指定する必要があります。プリンシパルには、アカウント、ユーザー、ロール、フェデレーションユーザー、または AWS のサービス を含めることができます。

クロスアカウントアクセスを有効にするには、全体のアカウント、または別のアカウントの IAM エンティティを、リソースベースのポリシーのプリンシパルとして指定します。リソースベースのポリシーにクロスアカウントのプリンシパルを追加しても、信頼関係は半分しか確立されない点に注意してください。プリンシパルとリソースが異なる AWS アカウント にある場合、信頼できるアカウントの IAM 管理者は、リソースにアクセスするための権限をプリンシパルエンティティ (ユーザーまたはロール) に付与する必要もあります。IAM 管理者は、アイデンティティベースのポリシーをエンティティにアタッチすることで権限を付与します。ただし、リソースベースのポリシーで、同じアカウントのプリンシパルへのアクセス権が付与されている場合は、アイデンティティベースのポリシーを追加する必要はありません。詳細については、『IAM ユーザーガイド』の「IAM ロールとリソースベースのポリシーとの相違点」を参照してください。

CloudWatch のポリシーアクション

ポリシーアクションに対するサポート	Yes

管理者は AWS JSON ポリシーを使用して、誰が何にアクセスできるかを指定できます。つまり、どのプリンシパルがどんなリソースにどんな条件でアクションを実行できるかということです。

JSON ポリシーのAction要素には、ポリシー内のアクセスを許可または拒否するために使用できるアクションが記述されます。ポリシーアクションの名前は通常、関連する AWSAPI オペレーションと同じです。一致する API オペレーションのない権限のみのアクションなど、いくつかの例外があります。また、ポリシーに複数アクションが必要なオペレーションもあります。これらの追加アクションは、 依存アクション と呼ばれます。

このアクションは、関連付けられたオペレーションを実行するための権限を付与するポリシーで使用されます。

CloudWatch アクションのリストを確認するには、「サービス認証リファレンス」の「Amazon CloudWatch で定義されるアクション」を参照してください。

CloudWatch のポリシーアクションは、アクションの前に次のプレフィックスを使用します。

```
cloudwatch
```

単一のステートメントで複数のアクションを指定するには、アクションをカンマで区切ります。

```
"Action": [
      "cloudwatch:action1",
      "cloudwatch:action2"
         ]
```

CloudWatch のアイデンティティベースのポリシー例については、「Amazon CloudWatch のアイデンティティベースのポリシー例」を参照してください。

CloudWatch のポリシーリソース

ポリシーリソースに対するサポート	Yes

管理者は AWS JSON ポリシーを使用して、誰が何にアクセスできるかを指定できます。つまり、どのプリンシパルがどのリソースにどのような条件でアクションを実行できるかということです。

Resource JSON ポリシー要素は、アクションが適用されるオブジェクトを指定します。ステートメントには、Resource または NotResource要素を含める必要があります。ベストプラクティスとして、Amazon リソースネーム (ARN) を使用してリソースを指定します。これは、リソースレベルの権限と呼ばれる特定のリソースタイプをサポートするアクションに対して実行できます。

オペレーションのリスト化など、リソースレベルの権限をサポートしないアクションの場合は、ステートメントがすべてのリソースに適用されることを示すために、ワイルドカード (*) を使用します。

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

CloudWatch リソースのタイプとその ARN のリストを確認するには、「サービス認証リファレンス」の「Amazon CloudWatch で定義されるリソースタイプ」を参照してください。どのアクションで各リソースの ARN を指定できるかについては、「Amazon CloudWatch で定義されるアクション」を参照してください。

CloudWatch のアイデンティティベースのポリシー例については、「Amazon CloudWatch のアイデンティティベースのポリシー例」を参照してください。

CloudWatch のポリシー条件キー

サービス固有のポリシー条件キーのサポート	はい

管理者は AWS JSON ポリシーを使用して、誰が何にアクセスできるかを指定できます。つまり、どのプリンシパルがどんなリソースにどんな条件でアクションを実行できるかということです。

Condition 要素 (または Condition ブロック) を使用すると、ステートメントが有効な条件を指定できます。Condition 要素はオプションです。イコールや未満などの 条件演算子 を使用して条件式を作成することで、ポリシーの条件とリクエスト内の値を一致させることができます。

1 つのステートメントに複数の Condition 要素を指定するか、1 つの Condition 要素に複数のキーを指定すると、AWS は AND 論理演算子を使用してそれらを評価します。単一の条件キーに複数

の値を指定すると、AWS は OR 論理演算子を使用して条件を評価します。ステートメントの権限が付与される前にすべての条件が満たされる必要があります。

条件を指定する際にプレースホルダー変数も使用できます。例えば IAM ユーザーに、IAM ユーザー名がタグ付けされている場合のみリソースにアクセスできる権限を付与することができます。詳細については、『IAM ユーザーガイド』の「IAM ポリシーの要素: 変数およびタグ」を参照してください。

AWS はグローバル条件キーとサービス固有の条件キーをサポートしています。すべての AWS グローバル条件キーを確認するには、IAM ユーザーガイド の「AWS グローバル条件コンテキストキー」を参照してください。

CloudWatch の条件キーのリストを確認するには、「サービス認証リファレンス」の「Amazon CloudWatch の条件キー」を参照してください。どのアクションやリソースで条件キーを使用できるかについては、「Amazon CloudWatch で定義されるアクション」を参照してください。

CloudWatch のアイデンティティベースのポリシー例については、「Amazon CloudWatch のアイデンティティベースのポリシー例」を参照してください。

CloudWatch の ACL

ACL のサポート	No

アクセスコントロールリスト (ACL) は、どのプリンシパル (アカウントメンバー、ユーザー、またはロール) がリソースにアクセスするための権限を持つかを制御します。ACL はリソースベースのポリシーに似ていますが、JSON ポリシードキュメント形式は使用しません。

CloudWatch での ABAC

ABAC (ポリシー内のタグ) のサポート	部分的

属性ベースのアクセスコントロール (ABAC) は、属性に基づいて権限を定義する認可戦略です。AWS では、属性は タグ と呼ばれます。タグは、IAM エンティティ (ユーザーまたはロール)、および多数の AWS リソースにアタッチできます。エンティティとリソースのタグ付けは、ABAC の最初の手順です。その後、プリンシパルのタグがアクセスしようとしているリソースのタグと一致した場合に操作を許可するように ABAC ポリシーを設計します。

ABAC は、急成長する環境やポリシー管理が煩雑になる状況で役立ちます。

タグに基づいてアクセスを管理するには、aws:ResourceTag/*key-name*、aws:RequestTag/*key-name*、または aws:TagKeys の条件キーを使用して、ポリシーの条件要素でタグ情報を提供します。

サービスがすべてのリソースタイプに対して 3 つの条件キーすべてをサポートする場合、そのサービスの値ははいです。サービスが一部のリソースタイプに対してのみ 3 つの条件キーのすべてをサポートする場合、値は「部分的」になります。

ABAC の詳細については、『IAM ユーザーガイド』の「ABAC とは?」を参照してください。ABAC をセットアップするステップを説明するチュートリアルについては、「IAM ユーザーガイド」の「属性に基づくアクセスコントロール (ABAC) を使用する」を参照してください。

CloudWatch での一時的な認証情報の使用

一時的な認証情報のサポート	Yes

AWS のサービスには、一時認証情報を使用してサインインしても機能しないものがあります。一時的な認証情報を利用できる AWS のサービス を含めた詳細情報については、『IAM ユーザーガイド』の「IAM と連携する AWS のサービス」を参照してください。

ユーザー名とパスワード以外の方法で AWS Management Consoleにサインインする場合は、一時認証情報を使用していることになります。例えば、会社のシングルサインオン (SSO) リンクを使用して AWS にアクセスすると、そのプロセスは自動的に一時認証情報を作成します。また、ユーザーとしてコンソールにサインインしてからロールを切り替える場合も、一時的な認証情報が自動的に作成されます。ロールの切り替えに関する詳細については、「IAM ユーザーガイド」の「ロールへの切り替え (コンソール)」を参照してください。

一時認証情報は、AWS CLI または AWSAPI を使用して手動で作成できます。作成後、一時認証情報を使用して AWSにアクセスできるようになります。AWS は、長期的なアクセスキーを使用する代わりに、一時認証情報を動的に生成することをお勧めします。詳細については、「IAM の一時的セキュリティ認証情報」を参照してください。

CloudWatch のクロスサービスプリンシパル許可

フォワードアクセスセッション (FAS) をサポート	Yes

IAM ユーザーまたはロールを使用して AWS でアクションを実行するユーザーは、プリンシパルとみなされます。一部のサービスを使用する際に、アクションを実行してから、別のサービスの別のアクションを開始することがあります。FAS は、AWS のサービスを呼び出すプリンシパルの権限を、AWS のサービスのリクエストと合わせて使用し、ダウンストリームのサービスに対してリクエストを行います。FAS リクエストは、サービスが、完了するために他の AWS のサービス またはリソースとのやりとりを必要とするリクエストを受け取ったときにのみ行われます。この場合、両方のアクションを実行するためのアクセス許可が必要です。FAS リクエストを行う際のポリシーの詳細については、「転送アクセスセッション」を参照してください。

CloudWatch のサービスロール

サービスロールに対するサポート	あり

サービスロールとは、サービスがユーザーに代わってアクションを実行するために引き受ける IAM ロールです。IAM 管理者は、IAM 内からサービスロールを作成、変更、削除できます。詳細については、「IAM ユーザーガイド」の「AWS のサービス に権限を委任するロールの作成」を参照してください。

> **Warning**
>
> サービスロールのアクセス許可を変更すると、CloudWatch の機能が破損する可能性があります。CloudWatch が指示する場合以外は、サービスロールを編集しないでください。

Amazon CloudWatch のアイデンティティベースのポリシー例

デフォルトでは、ユーザーおよびロールには、CloudWatch リソースを作成または変更するアクセス許可がありません。また、AWS Management Console、AWS Command Line Interface (AWS CLI)、または AWS API を使用してタスクを実行することもできません。IAM 管理者は、リソースで必要なアクションを実行するための権限をユーザーに付与する IAM ポリシーを作成できます。その後、管理者はロールに IAM ポリシーを追加し、ユーザーはロールを引き受けることができます。

これらサンプルの JSON ポリシードキュメントを使用して、IAM アイデンティティベースのポリシーを作成する方法については、『IAM ユーザーガイド』の「IAM ポリシーの作成」を参照してください。

CloudWatch が定義するアクションとリソースタイプ (リソースタイプごとの ARN の形式を含む) の詳細については、「サービス認証リファレンス」の「Amazon CloudWatch のアクション、リソース、および条件キー」を参照してください。

トピック

- ポリシーのベストプラクティス
- CloudWatch コンソールの使用

ポリシーのベストプラクティス

アイデンティティベースのポリシーは、アカウント内で、CloudWatch リソースを作成、アクセス、または削除できるかどうかを決定します。これらのアクションを実行すると、AWS アカウント に料金が発生する可能性があります。アイデンティティベースポリシーを作成したり編集したりする際には、以下のガイドラインと推奨事項に従ってください:

- AWS マネージドポリシーを使用して開始し、最小特権の権限に移行する – ユーザーとワークロードへの権限の付与を開始するには、多くの一般的なユースケースのために権限を付与する AWS マネージドポリシーを使用します。これらは AWS アカウントで使用できます。ユースケースに応じた AWS カスタマーマネージドポリシーを定義することで、権限をさらに減らすことをお勧めします。詳細については、『IAM ユーザーガイド』の「AWS マネージドポリシー」または「AWS ジョブ機能の管理ポリシー」を参照してください。
- 最小特権を適用する – IAM ポリシーで権限を設定するときは、タスクの実行に必要な権限のみを付与します。これを行うには、特定の条件下で特定のリソースに対して実行できるアクションを定義します。これは、最小特権権限とも呼ばれています。IAM を使用して権限を適用する方法の詳細については、『IAM ユーザーガイド』の「IAM でのポリシーと権限」を参照してください。
- IAM ポリシーで条件を使用してアクセスをさらに制限する - ポリシーに条件を追加して、アクションやリソースへのアクセスを制限できます。例えば、ポリシー条件を記述して、すべてのリクエストを SSL を使用して送信するように指定できます。また、AWS CloudFormation などの特定の AWS のサービスを介して使用する場合、条件を使ってサービスアクションへのアクセス権を付与することもできます。詳細については、『IAM ユーザーガイド』の [IAM JSON policy elements: Condition] (IAM JSON ポリシー要素：条件) を参照してください。
- IAM Access Analyzer を使用して IAM ポリシーを検証し、安全で機能的な権限を確保する - IAM Access Analyzer は、新規および既存のポリシーを検証して、ポリシーが IAM ポリシー言語 (JSON) および IAM のベストプラクティスに準拠するようにします。IAM アクセスアナライザーは 100 を超えるポリシーチェックと実用的な推奨事項を提供し、安全で機能的なポリシーの作成をサ

ポートします。詳細については、『IAM ユーザーガイド』の「IAM Access Analyzer ポリシーの検証」を参照してください。

- 多要素認証 (MFA) を要求する – AWS アカウント で IAM ユーザーまたはルートユーザーを要求するシナリオがある場合は、セキュリティを強化するために MFA をオンにします。API オペレーションが呼び出されるときに MFA を必須にするには、ポリシーに MFA 条件を追加します。詳細については、『IAM ユーザーガイド』の「MFA 保護 API アクセスの設定」を参照してください。

IAM でのベストプラクティスの詳細については、『IAM ユーザーガイド』の「IAM でのセキュリティのベストプラクティス」を参照してください。

CloudWatch コンソールの使用

Amazon CloudWatch コンソールにアクセスするには、最小限のアクセス許可が必要です。これらのアクセス許可により、AWS アカウント の CloudWatch リソースの詳細を一覧表示できます。最小限必要なアクセス許可よりも制限が厳しいアイデンティティベースのポリシーを作成すると、そのポリシーを持つエンティティ (ユーザーまたはロール) ではコンソールが意図したとおりに機能しません。

AWS CLI または AWS API のみを呼び出すユーザーには、最小限のコンソール権限を付与する必要はありません。代わりに、実行しようとしている API オペレーションに一致するアクションのみへのアクセスを許可します。

ユーザーとロールが引き続き CloudWatch コンソールを使用できるようにするには、エンティティに CloudWatch *ConsoleAccess* または *ReadOnly* AWS マネージドポリシーもアタッチします。詳細については、「IAM ユーザーガイド」の「ユーザーへのアクセス許可の追加」を参照してください。

CloudWatch コンソールへのアクセス許可

CloudWatch コンソールを使用して作業するのに必要なフルセットのアクセス許可は以下に記載されています。これらの許可があれば、CloudWatch コンソールへの書き込みおよび読み取りのフルアクセスが可能になります。

- application-autoscaling:DescribeScalingPolicies
- autoscaling:DescribeAutoScalingGroups
- autoscaling:DescribePolicies
- cloudtrail:DescribeTrails
- cloudwatch:DeleteAlarms

- cloudwatch:DescribeAlarmHistory
- cloudwatch:DescribeAlarms
- cloudwatch:GetMetricData
- cloudwatch:GetMetricStatistics
- cloudwatch:ListMetrics
- cloudwatch:PutMetricAlarm
- cloudwatch:PutMetricData
- ec2:DescribeInstances
- ec2:DescribeTags
- ec2:DescribeVolumes
- es:DescribeElasticsearchDomain
- es:ListDomainNames
- events:DeleteRule
- events:DescribeRule
- events:DisableRule
- events:EnableRule
- events:ListRules
- events:PutRule
- iam:AttachRolePolicy
- iam:CreateRole
- iam:GetPolicy
- iam:GetPolicyVersion
- iam:GetRole
- iam:ListAttachedRolePolicies
- iam:ListRoles
- kinesis:DescribeStream
- kinesis:ListStreams
- lambda:AddPermission
- lambda:CreateFunction
- lambda:GetFunctionConfiguration

- lambda:ListAliases
- lambda:ListFunctions
- lambda:ListVersionsByFunction
- lambda:RemovePermission
- logs:CancelExportTask
- logs:CreateExportTask
- logs:CreateLogGroup
- logs:CreateLogStream
- logs:DeleteLogGroup
- logs:DeleteLogStream
- logs:DeleteMetricFilter
- logs:DeleteRetentionPolicy
- logs:DeleteSubscriptionFilter
- logs:DescribeExportTasks
- logs:DescribeLogGroups
- logs:DescribeLogStreams
- logs:DescribeMetricFilters
- logs:DescribeQueries
- logs:DescribeSubscriptionFilters
- logs:FilterLogEvents
- logs:GetLogGroupFields
- logs:GetLogRecord
- logs:GetLogEvents
- logs:GetQueryResults
- logs:PutMetricFilter
- logs:PutRetentionPolicy
- logs:PutSubscriptionFilter
- logs:StartQuery
- logs:StopQuery
- logs:TestMetricFilter

- s3:CreateBucket
- s3:ListBucket
- sns:CreateTopic
- sns:GetTopicAttributes
- sns:ListSubscriptions
- sns:ListTopics
- sns:SetTopicAttributes
- sns:Subscribe
- sns:Unsubscribe
- sqs:GetQueueAttributes
- sqs:GetQueueUrl
- sqs:ListQueues
- sqs:SetQueueAttributes
- swf:CreateAction
- swf:DescribeAction
- swf:ListActionTemplates
- swf:RegisterAction
- swf:RegisterDomain
- swf:UpdateAction

また、X-Ray トレースマップを表示するには、`AWSXrayReadOnlyAccess` が必要です。

Amazon CloudWatch のアイデンティティとアクセスのトラブルシューティング

以下の情報は、CloudWatch と IAM の使用時に発生する可能性がある一般的な問題の診断や修復に役立ちます。

トピック

CloudWatch でアクションを実行する権限がない

「I am not authorized to perform an action in Amazon Bedrock」というエラーが表示された場合、そのアクションを実行できるようにポリシーを更新する必要があります。

次のエラー例は、mateojackson IAM ユーザーがコンソールを使用して、ある *my-example-widget* リソースに関する詳細情報を表示しようとしたことを想定して、その際に必要なcloudwatch:*GetWidget* アクセス許可を持っていない場合に発生するものです。

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 cloudwatch:GetWidget on resource: my-example-widget
```

この場合、cloudwatch:*GetWidget* アクションを使用して *my-example-widget*リソースへのアクセスを許可するように、mateojackson ユーザーのポリシーを更新する必要があります。

サポートが必要な場合は、AWS 管理者に問い合わせてください。管理者とは、サインイン認証情報を提供した担当者です。

iam:PassRole を実行する権限がありません

iam:PassRole アクションを実行する権限がないというエラーが表示された場合は、ポリシーを更新して CloudWatch にロールを渡すことができるようにする必要があります。

一部の AWS のサービスでは、新しいサービスロールまたはサービスリンクロールを作成せずに、既存のロールをサービスに渡すことが許可されています。そのためには、サービスにロールを渡す権限が必要です。

以下の例に示すエラーは、marymajor という名前の IAM ユーザーがコンソールを使用して CloudWatch でアクションを実行しようとした場合に発生します。ただし、このアクションをサービスが実行するには、サービスロールから付与された権限が必要です。Mary には、ロールをサービスに渡す権限がありません。

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

この場合、Mary のポリシーを更新してメアリーに iam:PassRole アクションの実行を許可する必要があります。

サポートが必要な場合は、AWS 管理者に問い合わせてください。管理者とは、サインイン認証情報を提供した担当者です。

自分の AWS アカウント 外のユーザーに CloudWatch リソースへのアクセスを許可したい

他のアカウントのユーザーや組織外の人が、リソースにアクセスするために使用できるロールを作成できます。ロールの引き受けを委託するユーザーを指定できます。リソースベースのポリシーまたはアクセスコントロールリスト (ACL) をサポートするサービスの場合、それらのポリシーを使用して、リソースへのアクセスを付与できます。

詳細については、以下を参照してください。

- CloudWatch でこれらの機能がサポートされているかどうかを確認するには、「Amazon CloudWatch と IAM の連携方法」を参照してください。
- 所有している AWS アカウント全体のリソースへのアクセス権を提供する方法については、「IAM ユーザーガイド」の「所有している別の AWS アカウントへのアクセス権を IAM ユーザーに提供」を参照してください。
- サードパーティーの AWS アカウント にリソースへのアクセス権を提供する方法については、『IAM ユーザーガイド』の「第三者が所有する AWS アカウント へのアクセス権を付与する」を参照してください。
- ID フェデレーションを介してアクセスを提供する方法については、『IAM ユーザーガイド』の「外部で認証されたユーザー (ID フェデレーション) へのアクセス権限」を参照してください。
- クロスアカウントアクセスでのロールとリソースベースのポリシーの使用の違いの詳細については、『IAM ユーザーガイド』の「IAM ロールとリソースベースのポリシーとの相違点」を参照してください。

CloudWatch ダッシュボード許可の更新

2018 年 5 月 1 日に、AWS は CloudWatch ダッシュボードにアクセスするために必要な許可を変更しました。CloudWatch コンソールでダッシュボードにアクセスするには、ダッシュボード API オペレーションをサポートするために 2017 年に導入された新しいアクセス許可が必要になりました。

- [cloudwatch:GetDashboard]
- [cloudwatch:ListDashboards]
- [cloudwatch:PutDashboard]
- [cloudwatch:DeleteDashboards]

CloudWatch ダッシュボードにアクセスするには、次のいずれかが必要です。

- [AdministratorAccess] ポリシー。
- [CloudWatchFullAccess] ポリシー。
- これらの特定のアクセス許可の 1 つ以上を含むカスタムポリシー。
  - ダッシュボードを表示できるようにするには `cloudwatch:GetDashboard` と `cloudwatch:ListDashboards`。
  - ダッシュボードを作成または変更できるようにするには `cloudwatch:PutDashboard`。
  - `cloudwatch:DeleteDashboards`ダッシュボードを削除できるようにするには 。

ポリシーを使用して IAM ユーザーのアクセス許可を変更する方法の詳細については、「IAM ユーザーのアクセス許可の変更」を参照してください。

CloudWatch のアクセス許可の詳細については、「Amazon CloudWatch の許可リファレンス」を参照してください。

ダッシュボード API オペレーションの詳細については、Amazon CloudWatch API リファレンスの「PutDashboard」を参照してください。

AWSCloudWatch の マネージド (事前定義) ポリシー

AWS は、 によって作成され管理されるスタンドアロンの IAM ポリシーを提供することで、多くの一般的ユースケースに対応します。AWSこれらの AWS 管理ポリシーは、一般的ユースケースに必要なアクセス権限を付与することで、どの権限が必要なのかをユーザーが調査する必要をなくすことができます。詳細については、IAM ユーザーガイドの AWS管理ポリシーを参照してください。

アカウントのユーザーにアタッチ可能な以下の AWS マネージドポリシーは、CloudWatch に固有のものです。

トピック

- CloudWatchFullAccessV2
- CloudWatchFullAccess
- CloudWatchReadOnlyAccess
- CloudWatchActionsEC2Access
- CloudWatchAutomaticDashboardsAccess
- CloudWatchAgentServerPolicy
- CloudWatchAgentAdminPolicy
- CloudWatch クロスアカウントのオブザーバビリティの AWS マネージド (事前定義済み) ポリシー

- CloudWatch Synthetics の AWS マネージド (事前定義された) ポリシー
- Amazon CloudWatch RUM の AWS マネージド (事前定義) ポリシー
- CloudWatch Evidently の AWS マネージド (事前定義) ポリシー
- AWS Systems Manager Incident Manager の AWS マネージドポリシー

CloudWatchFullAccessV2

AWS に CloudWatchFullAccessV2 マネージド IAM ポリシーが追加されました。このポリシーは、CloudWatch のアクションとリソースへのフルアクセスを許可し、Amazon SNS や Amazon EC2 Auto Scaling などの他のサービスに付与されるアクセス許可の範囲をより適切に設定します。CloudWatchFullAccess を使用する代わりに、このポリシーを使用し始めることをお勧めします。AWS は近い将来に CloudWatchFullAccess を廃止する予定です。

これには、ユーザーが Application Signals の CloudWatch コンソールからすべての機能にアクセスできる `application-signals:` アクセス許可が含まれています。これには、このポリシーを持つユーザーが CloudWatch アラームに関連付けられた Auto Scaling アクションを表示できるようにするためのいくつかの `autoscaling:Describe` アクセス許可が含まれます。これには、このポリシーを持つユーザーが Amazon SNS トピックを取得して CloudWatch アラームに関連付けることができるように、いくつかの `sns` 権限が含まれています。これには、このポリシーを持つユーザーが CloudWatch に関連付けられたサービスリンクロールに関する情報を表示できるようにするための IAM 許可が含まれます。これには、このポリシーを持つユーザーがコンソールを使用して、CloudWatch のクロスアカウントオブザーバビリティでソースアカウントから共有されたデータを表示できるようにするための `oam:ListSinks` および `oam:ListAttachedLinks` 許可が含まれます。

これには `rum`、`synthetics`、`xray` の各アクセス許可が含まれているため、ユーザーは CloudWatch Synthetics、AWS X-Ray、CloudWatch RUM にフルアクセスできます。いずれも、CloudWatch サービスの下にあります。

CloudWatchFullAccessV2 の内容は次のとおりです。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "CloudWatchFullAccessPermissions",
            "Effect": "Allow",
            "Action": [
                "application-autoscaling:DescribeScalingPolicies",
```

```
                "application-signals:*",
                "autoscaling:DescribeAutoScalingGroups",
                "autoscaling:DescribePolicies",
                "cloudwatch:*",
                "logs:*",
                "sns:CreateTopic",
                "sns:ListSubscriptions",
                "sns:ListSubscriptionsByTopic",
                "sns:ListTopics",
                "sns:Subscribe",
                "iam:GetPolicy",
                "iam:GetPolicyVersion",
                "iam:GetRole",
                "oam:ListSinks",
                "rum:*",
                "synthetics:*",
                "xray:*"
            ],
            "Resource": "*"
        },
        {
            "Sid": "CloudWatchApplicationSignalsServiceLinkedRolePermissions",
            "Effect": "Allow",
            "Action": "iam:CreateServiceLinkedRole",
            "Resource": "arn:aws:iam::*:role/aws-service-role/application-
signals.cloudwatch.amazonaws.com/AWSServiceRoleForCloudWatchApplicationSignals",
            "Condition": {
                "StringLike": {
                    "iam:AWSServiceName": "application-
signals.cloudwatch.amazonaws.com"
                }
            }
        },
        {
            "Sid": "EventsServicePermissions",
            "Effect": "Allow",
            "Action": "iam:CreateServiceLinkedRole",
            "Resource": "arn:aws:iam::*:role/aws-service-role/events.amazonaws.com/
AWSServiceRoleForCloudWatchEvents*",
            "Condition": {
                "StringLike": {
                    "iam:AWSServiceName": "events.amazonaws.com"
                }
            }
```

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

CloudWatchFullAccess

CloudWatchFullAccess ポリシーは、近い将来に廃止されます。このポリシーの使用を中止し、代わりに CloudWatchFullAccessV2 を使用することをお勧めします。

CloudWatchFullAccess の内容は次のとおりです。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "autoscaling:Describe*",
                "cloudwatch:*",
                "logs:*",
                "sns:*",
                "iam:GetPolicy",
                "iam:GetPolicyVersion",
                "iam:GetRole",
                "oam:ListSinks"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": "iam:CreateServiceLinkedRole",
            "Resource": "arn:aws:iam::*:role/aws-service-role/events.amazonaws.com/
AWSServiceRoleForCloudWatchEvents*",
            "Condition": {
                "StringLike": {
```

```
                    "iam:AWSServiceName": "events.amazonaws.com"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "oam:ListAttachedLinks"
            ],
            "Resource": "arn:aws:oam:*:*:sink/*"
        }
    ]
}
```

CloudWatchReadOnlyAccess

CloudWatchReadOnlyAccess ポリシーは、CloudWatch への読み取り専用のアクセス権限を付与します。

このポリシーには logs: アクセス許可がいくつか含まれているため、このポリシーを持つユーザーはコンソールを使用して CloudWatch Logs 情報と CloudWatch Logs Insights クエリを表示できます。これには autoscaling:Describe* が含まれているため、このポリシーを持つユーザーは CloudWatch アラームに関連付けられた Auto Scaling アクションを確認できます。これには、ユーザーが Application Signals を使用してサービスの状態をモニタリングできるようにする application-signals: アクセス許可が含まれています。これには application-autoscaling:DescribeScalingPolicies が含まれているため、このポリシーを持つユーザーは Application Auto Scaling ポリシーに関する情報にアクセスできます。これには sns:Get* と sns:List* が含まれているため、このポリシーを持つユーザーはどの Amazon SNS トピックを取得すれば CloudWatch アラームに関する通知が届くのか、必要な情報を得ることができます。これには oam:ListSinks と oam:ListAttachedLinks の各アクセス許可が含まれているため、このポリシーを持つユーザーはコンソールを使用して CloudWatch のクロスアカウントオブザーバビリティでソースアカウントから共有されたデータを表示できます。これには、ユーザーが CloudWatch Application Signals がセットアップされているかどうかを確認するための iam:GetRole アクセス許可が含まれています。

これには rum、synthetics、xray の各アクセス許可が含まれているため、ユーザーは CloudWatch Synthetics、AWS X-Ray、CloudWatch RUM に読み取り専用でアクセスできます。いずれも、CloudWatch サービスの下にあります。

CloudWatchReadOnlyAccess ポリシーの内容は次のとおりです。

```
{
"Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "CloudWatchReadOnlyAccessPermissions",
            "Effect": "Allow",
            "Action": [
                "application-autoscaling:DescribeScalingPolicies",
                "application-signals:BatchGet*",
                "application-signals:Get*",
                "application-signals:List*",
                "autoscaling:Describe*",
                "cloudwatch:BatchGet*",
                "cloudwatch:Describe*",
                "cloudwatch:GenerateQuery",
                "cloudwatch:Get*",
                "cloudwatch:List*",
                "logs:Get*",
                "logs:List*",
                "logs:StartQuery",
                "logs:StopQuery",
                "logs:Describe*",
                "logs:TestMetricFilter",
                "logs:FilterLogEvents",
                "logs:StartLiveTail",
                "logs:StopLiveTail",
                "oam:ListSinks",
                "sns:Get*",
                "sns:List*",
                "rum:BatchGet*",
                "rum:Get*",
                "rum:List*",
                "synthetics:Describe*",
                "synthetics:Get*",
                "synthetics:List*",
                "xray:BatchGet*",
                "xray:Get*"
            ],
            "Resource": "*"
        },
        {
            "Sid": "OAMReadPermissions",
            "Effect": "Allow",
```

```
            "Action": [
                "oam:ListAttachedLinks"
            ],
            "Resource": "arn:aws:oam:*:*:sink/*"
        },
        {
            "Sid": "CloudWatchReadOnlyGetRolePermissions",
            "Effect": "Allow",
            "Action": "iam:GetRole",
            "Resource": "arn:aws:iam::*:role/aws-service-role/application-
signals.cloudwatch.amazonaws.com/AWSServiceRoleForCloudWatchApplicationSignals"
        }
    ]
}
```

CloudWatchActionsEC2Access

CloudWatchActionsEC2Access ポリシーは、Amazon EC2 メタデータに加えて CloudWatch アラームとメトリクスに対し、読み取り専用のアクセス許可を付与します。また、EC2 インスタンス用の API アクションに停止、終了、再起動のためのアクセス権限を付与します。

CloudWatchActionsEC2Access ポリシーの内容は次のとおりです。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "cloudwatch:Describe*",
        "ec2:Describe*",
        "ec2:RebootInstances",
        "ec2:StopInstances",
        "ec2:TerminateInstances"
      ],
      "Resource": "*"
    }
  ]
}
```

CloudWatchAutomaticDashboardsAccess

CloudWatch-CrossAccountAccess 管理ポリシーは、CloudWatch-CrossAccountSharingRole IAM ロールにより使用されます。このロールとポリシーを使用することで、クロスアカウントダッシュボードのユーザーが、ダッシュボードを共有している各アカウントにおいて、自動ダッシュボードを表示できるようになります。

次に、CloudWatchAutomaticDashboardsAccess の内容を示します。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "autoscaling:DescribeAutoScalingGroups",
        "cloudfront:GetDistribution",
        "cloudfront:ListDistributions",
        "dynamodb:DescribeTable",
        "dynamodb:ListTables",
        "ec2:DescribeInstances",
        "ec2:DescribeVolumes",
        "ecs:DescribeClusters",
        "ecs:DescribeContainerInstances",
        "ecs:ListClusters",
        "ecs:ListContainerInstances",
        "ecs:ListServices",
        "elasticache:DescribeCacheClusters",
        "elasticbeanstalk:DescribeEnvironments",
        "elasticfilesystem:DescribeFileSystems",
        "elasticloadbalancing:DescribeLoadBalancers",
        "kinesis:DescribeStream",
        "kinesis:ListStreams",
        "lambda:GetFunction",
        "lambda:ListFunctions",
        "rds:DescribeDBClusters",
        "rds:DescribeDBInstances",
        "resource-groups:ListGroupResources",
        "resource-groups:ListGroups",
        "route53:GetHealthCheck",
        "route53:ListHealthChecks",
        "s3:ListAllMyBuckets",
        "s3:ListBucket",
        "sns:ListTopics",
```

```
        "sqs:GetQueueAttributes",
        "sqs:GetQueueUrl",
        "sqs:ListQueues",
        "synthetics:DescribeCanariesLastRun",
        "tag:GetResources"
      ],
      "Effect": "Allow",
      "Resource": "*"
    },
    {
      "Action": [
        "apigateway:GET"
      ],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:apigateway:*::/restapis*"
      ]
    }
  ]
```

CloudWatchAgentServerPolicy

CloudWatchAgentServerPolicy ポリシーは、Amazon EC2 インスタンスにアタッチされた IAM ロールで使用でき、CloudWatch エージェントがインスタンスから情報を読み取り、CloudWatch に書き込むことを許可します。内容は次のとおりです。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "CWACloudWatchServerPermissions",
            "Effect": "Allow",
            "Action": [
                "cloudwatch:PutMetricData",
                "ec2:DescribeVolumes",
                "ec2:DescribeTags",
                "logs:PutLogEvents",
                "logs:PutRetentionPolicy",
                "logs:DescribeLogStreams",
                "logs:DescribeLogGroups",
                "logs:CreateLogStream",
                "logs:CreateLogGroup",
                "xray:PutTraceSegments",
```

```
                "xray:PutTelemetryRecords",
                "xray:GetSamplingRules",
                "xray:GetSamplingTargets",
                "xray:GetSamplingStatisticSummaries"
            ],
            "Resource": "*"
        },
        {
            "Sid": "CWASSMServerPermissions",
            "Effect": "Allow",
            "Action": [
                "ssm:GetParameter"
            ],
            "Resource": "arn:aws:ssm:*:*:parameter/AmazonCloudWatch-*"
        }
    ]
}
```

CloudWatchAgentAdminPolicy

CloudWatchAgentAdminPolicy ポリシーは、Amazon EC2 インスタンスにアタッチされた IAM ロールで使用できます。このポリシーにより、CloudWatch エージェントはインスタンスから情報を読み取って CloudWatch に書き込み、パラメータストアに情報を書き込むことができます。内容は次のとおりです。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "CWACloudWatchPermissions",
            "Effect": "Allow",
            "Action": [
                "cloudwatch:PutMetricData",
                "ec2:DescribeTags",
                "logs:PutLogEvents",
                "logs:PutRetentionPolicy",
                "logs:DescribeLogStreams",
                "logs:DescribeLogGroups",
                "logs:CreateLogStream",
                "logs:CreateLogGroup",
                "xray:PutTraceSegments",
                "xray:PutTelemetryRecords",
                "xray:GetSamplingRules",
```

```
                "xray:GetSamplingTargets",
                "xray:GetSamplingStatisticSummaries"
            ],
            "Resource": "*"
        },
        {
            "Sid": "CWASSMPermissions",
            "Effect": "Allow",
            "Action": [
                "ssm:GetParameter",
                "ssm:PutParameter"
            ],
            "Resource": "arn:aws:ssm:*:*:parameter/AmazonCloudWatch-*"
        }
    ]
}
```

> **Note**
>
> IAM コンソールにサインインし、特定のポリシーを検索することで、これらのアクセス許可ポリシーを確認できます。

独自のカスタム IAM ポリシーを作成して、CloudWatch アクションとリソースのためのアクセス権限を許可することもできます。これらのカスタムポリシーは、それらのアクセス許可が必要な IAM ユーザーまたはグループにアタッチできます。

CloudWatch クロスアカウントのオブザーバビリティの AWS マネージド (事前定義済み) ポリシー

このセクションのポリシーは、CloudWatch クロスアカウントオブザーバビリティに関連する許可を付与します。詳細については、「CloudWatch のクロスアカウントオブザーバビリティ」を参照してください。

CloudWatchCrossAccountSharingConfiguration

CloudWatchCrossAccountSharingConfiguration ポリシーは、アカウント間で CloudWatch リソースを共有するための Observability Access Manager リンクを作成、管理、および表示するためのアクセス権を付与します。詳細については、「CloudWatch のクロスアカウントオブザーバビリティ」を参照してください。内容は次のとおりです。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "cloudwatch:Link",
                "oam:ListLinks"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "oam:DeleteLink",
                "oam:GetLink",
                "oam:TagResource"
            ],
            "Resource": "arn:aws:oam:*:*:link/*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "oam:CreateLink",
                "oam:UpdateLink"
            ],
            "Resource": [
                "arn:aws:oam:*:*:link/*",
                "arn:aws:oam:*:*:sink/*"
            ]
        }
    ]
}
```

OAMFullAccess

OAMFullAccess ポリシーは、CloudWatch クロスアカウントオブザーバビリティ用に使用される Observability Access Manager のシンクとリンクを作成、管理、および表示するためのアクセス権を付与します。

OAMFullAccess ポリシーだけでは、リンク間でのオブザーバビリティデータの共有は許可されません。CloudWatch メトリクスを共有するためのリンクを作成するには、CloudWatchFullAccess

または CloudWatchCrossAccountSharingConfiguration も必要になります。CloudWatch Logs ロググループを共有するためのリンクを作成するには、CloudWatchLogsFullAccess または CloudWatchLogsCrossAccountSharingConfiguration も必要になります。X-Ray トレースを共有するためのリンクを作成するには、AWSXRayFullAccess または AWSXRayCrossAccountSharingConfiguration も必要になります。

詳細については、「CloudWatch のクロスアカウントオブザーバビリティ」を参照してください。内容は次のとおりです。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "oam:*"
            ],
            "Resource": "*"
        }
    ]
}
```

OAMReadOnlyAccess

OAMReadOnlyAccess ポリシーは、CloudWatch クロスアカウントオブザーバビリティ用に使用される Observability Access Manager リソースへの読み取り専用アクセス権を付与します。詳細については、「CloudWatch のクロスアカウントオブザーバビリティ」を参照してください。内容は次のとおりです。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "oam:Get*",
                "oam:List*"
            ],
            "Resource": "*"
        }
    ]
```

```
}
```

CloudWatch Synthetics の AWS マネージド (事前定義された) ポリシー

CloudWatchSyntheticsFullAccess および CloudWatchSyntheticsReadOnlyAccess AWS マネージドポリシーは、CloudWatch Synthetics を管理または使用するユーザーに割り当てることができます。次の追加ポリシーも関連します。

- AmazonS3ReadOnlyAccess および CloudWatchReadOnlyAccess – これらは、CloudWatch コンソールですべての Synthetics データを読み取ることができるようにするために必要です。
- [AWSLambdaReadOnlyAccess] – Canary で使用されているソースコードを表示できるようにします。
- [CloudWatchSyntheticsFullAccess] で、Canary を作成できるようになります。さらに、Canary 用に作成された新しい IAM ロールを持つ Canary の作成と削除には、次のインラインポリシーステートメントも必要です。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iam:CreateRole",
                "iam:DeleteRole",
                "iam:CreatePolicy",
                "iam:DeletePolicy",
                "iam:AttachRolePolicy",
                "iam:DetachRolePolicy",
            ],
            "Resource": [
                "arn:aws:iam::*:role/service-role/CloudWatchSyntheticsRole*",
                "arn:aws:iam::*:policy/service-role/CloudWatchSyntheticsPolicy*"
            ]
        }
    ]
}
```

> **Important**
>
> ユーザーに iam:CreateRole、iam:DeleteRole、iam:CreatePolicy、iam:DeletePolicy、iam:Atta および iam:DetachRolePolicy のアクセス許可を付与すると、arn:aws:iam::*:role/service-role/CloudWatchSyntheticsRole* および arn:aws:iam::*:policy/service-role/CloudWatchSyntheticsPolicy* に一致する ARN を持つロールおよびポリシーを作成、アタッチ、および削除する完全な管理アクセス権がそのユーザーに付与されます。例えば、これらのアクセス許可を持つユーザーは、すべてのリソースに対する完全なアクセス許可を持つポリシーを作成し、そのポリシーを上記の ARN パターンに一致する任意のロールにアタッチできます。これらのアクセス許可を付与するユーザーには十分注意してください。

ポリシーのアタッチとユーザーへのアクセス許可の付与については、「IAM ユーザーのアクセス許可の変更」および「ユーザーまたはロールのインラインポリシーを埋め込むには」を参照してください。

CloudWatchSyntheticsFullAccess

CloudWatchSyntheticsFullAccess ポリシーの内容は次のとおりです。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "synthetics:*"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:CreateBucket",
                "s3:PutEncryptionConfiguration"
            ],
            "Resource": [
```

```
                "arn:aws:s3:::cw-syn-results-*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iam:ListRoles",
                "s3:ListAllMyBuckets",
                "xray:GetTraceSummaries",
                "xray:BatchGetTraces",
                "apigateway:GET"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetBucketLocation"
            ],
            "Resource": "arn:aws:s3:::*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:ListBucket"
            ],
            "Resource": "arn:aws:s3:::cw-syn-*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObjectVersion"
            ],
            "Resource": "arn:aws:s3:::aws-synthetics-library-*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "iam:PassRole"
            ],
            "Resource": [
                "arn:aws:iam::*:role/service-role/CloudWatchSyntheticsRole*"
            ],
```

```
            "Condition": {
                "StringEquals": {
                    "iam:PassedToService": [
                        "lambda.amazonaws.com",
                        "synthetics.amazonaws.com"
                    ]
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "iam:GetRole",
                "iam:ListAttachedRolePolicies"
            ],
            "Resource": [
                "arn:aws:iam::*:role/service-role/CloudWatchSyntheticsRole*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "cloudwatch:GetMetricData",
                "cloudwatch:GetMetricStatistics"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "cloudwatch:PutMetricAlarm",
                "cloudwatch:DeleteAlarms"
            ],
            "Resource": [
                "arn:aws:cloudwatch:*:*:alarm:Synthetics-*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "cloudwatch:DescribeAlarms"
            ],
            "Resource": [
                "arn:aws:cloudwatch:*:*:alarm:*"
```

```
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "lambda:CreateFunction",
                "lambda:AddPermission",
                "lambda:PublishVersion",
                "lambda:UpdateFunctionCode",
                "lambda:UpdateFunctionConfiguration",
                "lambda:GetFunctionConfiguration",
                "lambda:DeleteFunction"
            ],
            "Resource": [
                "arn:aws:lambda:*:*:function:cwsyn-*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "lambda:GetLayerVersion",
                "lambda:PublishLayerVersion",
                "lambda:DeleteLayerVersion"
            ],
            "Resource": [
                "arn:aws:lambda:*:*:layer:cwsyn-*",
                "arn:aws:lambda:*:*:layer:Synthetics:*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "ec2:DescribeVpcs",
                "ec2:DescribeSubnets",
                "ec2:DescribeSecurityGroups"
            ],
            "Resource": [
                "*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "sns:ListTopics"
```

```
            ],
            "Resource": [
                "*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "sns:CreateTopic",
                "sns:Subscribe",
                "sns:ListSubscriptionsByTopic"
            ],
            "Resource": [
                "arn:*:sns:*:*:Synthetics-*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "kms:ListAliases"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "kms:DescribeKey"
            ],
            "Resource": "arn:aws:kms:*:*:key/*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "kms:Decrypt"
            ],
            "Resource": "arn:aws:kms:*:*:key/*",
            "Condition": {
                "StringLike": {
                    "kms:ViaService": [
                        "s3.*.amazonaws.com"
                    ]
                }
            }
        }
```

```
    ]
}
```

CloudWatchSyntheticsReadOnlyAccess

CloudWatchSyntheticsReadOnlyAccess ポリシーの内容は次のとおりです。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "synthetics:Describe*",
                "synthetics:Get*",
                "synthetics:List*",
                "lambda:GetFunctionConfiguration"
            ],
            "Resource": "*"
        }
    ]
}
```

Amazon CloudWatch RUM の AWS マネージド (事前定義) ポリシー

AmazonCloudWatchRUMFullAccess および AmazonCloudWatchRUMReadOnlyAccess AWS マネージドポリシーは、CloudWatch RUM を管理または使用するユーザーに割り当てることができます。

AmazonCloudWatchRUMFullAccess

AmazonCloudWatchRUMFullAccess ポリシーの内容は次のとおりです。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "rum:*"
            ],
            "Resource": "*"
        },
        {
```

```
            "Effect": "Allow",
            "Action": [
                "iam:GetRole",
                "iam:CreateServiceLinkedRole"
            ],
            "Resource": [
                "arn:aws:iam::*:role/aws-service-role/rum.amazonaws.com/
AWSServiceRoleForRealUserMonitoring"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iam:PassRole"
            ],
            "Resource": [
                "arn:aws:iam::*:role/RUM-Monitor*"
            ],
            "Condition": {
                "StringEquals": {
                    "iam:PassedToService": [
                        "cognito-identity.amazonaws.com"
                    ]
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "cloudwatch:GetMetricData",
                "cloudwatch:GetMetricStatistics",
                "cloudwatch:ListMetrics"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "cloudwatch:DescribeAlarms"
            ],
            "Resource": "arn:aws:cloudwatch:*:*:alarm:*"
        },
        {
            "Effect": "Allow",
```

```
            "Action": [
                "cognito-identity:CreateIdentityPool",
                "cognito-identity:ListIdentityPools",
                "cognito-identity:DescribeIdentityPool",
                "cognito-identity:GetIdentityPoolRoles",
                "cognito-identity:SetIdentityPoolRoles"
            ],
            "Resource": "arn:aws:cognito-identity:*:*:identitypool/*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogGroup",
                "logs:DeleteLogGroup",
                "logs:PutRetentionPolicy",
                "logs:CreateLogStream"
            ],
            "Resource": "arn:aws:logs:*:*:log-group:*RUMService*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogDelivery",
                "logs:GetLogDelivery",
                "logs:UpdateLogDelivery",
                "logs:DeleteLogDelivery",
                "logs:ListLogDeliveries",
                "logs:DescribeResourcePolicies"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "logs:DescribeLogGroups"
            ],
            "Resource": "arn:aws:logs:*:*:log-group::log-stream:*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "synthetics:describeCanaries",
                "synthetics:describeCanariesLastRun"
            ],
```

```
            "Resource": "arn:aws:synthetics:*:*:canary:*"
        }
    ]
}
```

AmazonCloudWatchRUMReadOnlyAccess

AmazonCloudWatchRUMReadOnlyAccess ポリシーの内容は次のとおりです。

```
{

 "Version": "2012-10-17",
 "Statement": [
  {
   "Effect": "Allow",
   "Action": [
    "rum:GetAppMonitor",
    "rum:GetAppMonitorData",
    "rum:ListAppMonitors",
    "rum:ListRumMetricsDestinations",
    "rum:BatchGetRumMetricDefinitions"
   ],
   "Resource": "*"
  }
 ]
}
```

AmazonCloudWatchRUMServiceRolePolicy

AmazonCloudWatchRUMServiceRolePolicy は IAM エンティティにアタッチできません。このポリシーは、モニターリングデータを他の関連する AWS のサービスに CloudWatch RUM が公開することを許可する、サービスにリンクされたロールにアタッチされます。サービスにリンクされたこのロールの詳細については、「CloudWatch RUM のサービスにリンクされたロールの使用」を参照してください。

AmazonCloudWatchRUMServiceRolePolicy の詳細な内容は次のとおりです。

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

```
    "xray:PutTraceSegments"
   ],
   "Resource": [
    "*"
   ]
  },
  {
   "Effect": "Allow",
   "Action": "cloudwatch:PutMetricData",
   "Resource": "*",
   "Condition": {
    "StringLike": {
     "cloudwatch:namespace": [
      "RUM/CustomMetrics/*",
      "AWS/RUM"
     ]
    }
   }
  }
 ]
}
```

CloudWatch Evidently の AWS マネージド (事前定義) ポリシー

CloudWatchEvidentlyFullAccess および CloudWatchEvidentlyReadOnlyAccess AWS マネージドポリシーは、CloudWatch Evidently を管理または使用するユーザーに割り当てることができます。

CloudWatchEvidentlyFullAccess

CloudWatchEvidentlyFullAccess ポリシーの内容は次のとおりです。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "evidently:*"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
```

```
            "Action": [
                "iam:ListRoles"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "iam:GetRole"
            ],
            "Resource": [
                "arn:aws:iam::*:role/service-role/CloudWatchRUMEvidentlyRole-*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetBucketLocation",
                "s3:ListAllMyBuckets"
            ],
            "Resource": "arn:aws:s3:::*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "cloudwatch:GetMetricData",
                "cloudwatch:GetMetricStatistics",
                "cloudwatch:DescribeAlarmHistory",
                "cloudwatch:DescribeAlarmsForMetric",
                "cloudwatch:ListTagsForResource"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "cloudwatch:DescribeAlarms",
                "cloudwatch:TagResource",
                "cloudwatch:UnTagResource"
            ],
            "Resource": [
                "arn:aws:cloudwatch:*:*:alarm:*"
            ]
        },
```

```
        {
            "Effect": "Allow",
            "Action": [
                "cloudtrail:LookupEvents"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "cloudwatch:PutMetricAlarm"
            ],
            "Resource": [
                "arn:aws:cloudwatch:*:*:alarm:Evidently-Alarm-*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "sns:ListTopics"
            ],
            "Resource": [
                "*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "sns:CreateTopic",
                "sns:Subscribe",
                "sns:ListSubscriptionsByTopic"
            ],
            "Resource": [
                "arn:*:sns:*:*:Evidently-*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "logs:DescribeLogGroups"
            ],
            "Resource": [
                "*"
            ]
```

```
        }
    ]
}
```

CloudWatchEvidentlyReadOnlyAccess

CloudWatchEvidentlyReadOnlyAccess ポリシーの内容は次のとおりです。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "evidently:GetExperiment",
                "evidently:GetFeature",
                "evidently:GetLaunch",
                "evidently:GetProject",
                "evidently:GetSegment",
                "evidently:ListExperiments",
                "evidently:ListFeatures",
                "evidently:ListLaunches",
                "evidently:ListProjects",
                "evidently:ListSegments",
                "evidently:ListSegmentReferencs"
            ],
            "Resource": "*"
        }
    ]
}
```

AWS Systems Manager Incident Manager の AWS マネージドポリシー

AWSCloudWatchAlarms_ActionSSMIncidentsServiceRolePolicy ポリシーは、サービスにリンクされたロールにアッタッチされ、CloudWatch がお客様に代わって AWS Systems Manager Incident Manager でインシデントを開始できるようにします。詳細については、「CloudWatch アラーム Systems Manager Incident Manager アクションのサービスにリンクされたロールの許可」を参照してください。

このポリシーには、以下のアクセス許可があります。

- ssm-incidents:StartIncident

お客様が管理するポリシーの例

このセクションでは、さまざまな CloudWatch アクションのアクセス権限を付与するユーザーポリシー例を示しています。これらのポリシーは、CloudWatch API、AWS SDK、または AWS CLI を使用しているときに機能します。

例

例 1: ユーザーに CloudWatch への完全アクセスを許可する

CloudWatch へのフルアクセスをユーザーに付与するには、カスタマー管理ポリシーを作成する代わりに、CloudWatchFullAccess 管理ポリシーをユーザーに付与できます。CloudWatchFullAccess の内容を CloudWatchFullAccess に示します。

例 2: CloudWatch への読み取り専用アクセスを許可する

次のポリシーにより、ユーザーは CloudWatch への読み取り専用アクセスを行え、Amazon EC2 Auto Scaling アクション、CloudWatch メトリクス、CloudWatch Logs データ、アラーム関連の Amazon SNS データを表示できます。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "autoscaling:Describe*",
        "cloudwatch:Describe*",
        "cloudwatch:Get*",
        "cloudwatch:List*",
        "logs:Get*",
        "logs:Describe*",
        "logs:StartQuery",
        "logs:StopQuery",
        "logs:TestMetricFilter",
        "logs:FilterLogEvents",
        "logs:StartLiveTail",
        "logs:StopLiveTail",
```

```
        "sns:Get*",
        "sns:List*"
      ],
      "Effect": "Allow",
      "Resource": "*"
    }
  ]
}
```

例 3: Amazon EC2 インスタンスを停止または終了する

以下のポリシーは、CloudWatch アラームアクションにより EC2 インスタンスの停止または終了を許可します。下の例では、GetMetricData、ListMetrics、DescribeAlarms アクションはオプションです。これらのアクションを含めて、インスタンスが正常に停止または終了したことを確認することをお勧めします。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "cloudwatch:PutMetricAlarm",
        "cloudwatch:GetMetricData",
        "cloudwatch:ListMetrics",
        "cloudwatch:DescribeAlarms"
      ],
      "Resource": [
        "*"
      ],
      "Effect": "Allow"
    },
    {
      "Action": [
        "ec2:DescribeInstanceStatus",
        "ec2:DescribeInstances",
        "ec2:StopInstances",
        "ec2:TerminateInstances"
      ],
      "Resource": [
        "*"
      ],
      "Effect": "Allow"
    }
```

```
  ]
}
```

CloudWatch の AWS マネージドポリシーへの更新

CloudWatch の AWS マネージドポリシーの更新に関する詳細を、このサービスがこれらの変更の追跡を開始した以降の分について表示します。このページの変更に関する自動通知を入手するには、CloudWatch ドキュメントの履歴ページから、RSS フィードにサブスクライブしてください。

変更	説明	日付
CloudWatchFullAccessV2 – 既存のポリシーに対する更新	CloudWatch で、CloudWatchFullAccessV2 という名前のポリシーが更新されました。 `CloudWatchFullAccessPermissions` ポリシーのスコープが更新されて `application-signals:*` が追加され、ユーザーは、CloudWatch Application Signals を使ってサービスの正常性を表示、調査、診断できるようになりました。	2024 年 5 月 20 日
CloudWatchReadOnlyAccess – 既存のポリシーに対する更新	CloudWatch で、CloudWatchReadOnlyAccess という名前のポリシーが更新されました。 `CloudWatchReadOnlyAccessPermissions` ポリシーのスコープが更新されて `application-signals:BatchGet*` 、`application-signal`	2024 年 5 月 20 日

変更	説明	日付
	s:List* 、application-signals:Get* が追加され、ユーザーは、CloudWatch Application Signals を使ってサービスの正常性を表示、調査、診断できるようになりました。CloudWatchReadOnlyGetRolePermissions のスコープが更新されて iam:GetRole アクションが追加され、ユーザーは、CloudWatch Application Signals がセットアップされているかどうかをチェックできるようになりました。	
CloudWatchApplicationSignalsServiceRolePolicy – 既存のポリシーに対する更新	CloudWatch では、CloudWatchApplicationSignalsServiceRolePolicy という名前のポリシーが更新されました。 logs:StartQuery および logs:GetQueryResults のアクセス許可のスコープが変更され、arn:aws:logs:*:*:log-group:/aws/appsignals/*:* および arn:aws:logs:*:*:log-group:/aws/application-signals/data:* ARN が追加され、より多くのアーキテクチャで Application Signals を有効にできるようになりました。	2024 年 4 月 18 日

変更	説明	日付
CloudWatchApplicationSignalsServiceRolePolicy – 既存のポリシーに対する更新	CloudWatch では、CloudWatchApplicationSignalsServiceRolePolicy のアクセス許可のスコープが変更されました。 `cloudwatch:GetMetricData` アクセス許可のスコープが * に変更され、Application Signals が連結アカウントのソースからメトリクスを取得できるようになりました。	2024 年 4 月 8 日
CloudWatchAgentServerPolicy – 既存のポリシーに対する更新	CloudWatch が、CloudWatchAgentServerPolicy に対するアクセス許可を追加しました。 `xray:PutTraceSegments` 、`xray:PutTelemetryRecords` 、`xray:GetSamplingRules` 、`xray:GetSamplingTargets` 、`xray:GetSamplingStatisticSummaries` および `logs:PutRetentionPolicy` のアクセス許可が追加され、CloudWatch エージェントが X-Ray トレースを公開し、ロググループの保持期間を変更できるようになりました。	2024 年 2 月 21 日

変更	説明	日付
CloudWatchAgentAdminPolicy – 既存のポリシーに対する更新	CloudWatch が、CloudWatchAgentAdminPolicy に対するアクセス許可を追加しました。 `xray:PutTraceSegments` 、`xray:PutTelemetryRecords` 、`xray:GetSamplingRules` 、`xray:GetSamplingTargets` 、`xray:GetSamplingStatisticSummaries` および `logs:PutRetentionPolicy` のアクセス許可が追加され、CloudWatch エージェントが X-Ray トレースを公開し、ロググループの保持期間を変更できるようになりました。	2024 年 2 月 21 日

変更	説明	日付
CloudWatchFullAccessV2 – 既存のポリシーに対する更新	CloudWatchFullAccessV2 に対するアクセス許可を追加しました。 このポリシーを持つユーザーが CloudWatch Application Signals を管理できるように、CloudWatch Synthetics、X-Ray、CloudWatch RUM の各アクションに対する既存のアクセス許可と、CloudWatch Application Signals に対する新しいアクセス許可を追加しました。 CloudWatch Application Signals がログ、メトリクス、トレース、タグ内のテレメトリデータを検出できるように、CloudWatch Application Signals サービスリンクロールを作成するためのアクセス許可を追加しました。	2023 年 12 月 5 日

変更	説明	日付
CloudWatchReadOnlyAccess – 既存のポリシーに対する更新	CloudWatch が、CloudWatchReadOnlyAccess に対する許可を追加しました。 このポリシーを持つユーザーが CloudWatch Application Signals から報告されたサービス状態の問題をトリアージして診断できるように、CloudWatch Synthetics、X-Ray、CloudWatch RUM の各アクションに対する既存の読み取り専用アクセス許可と、CloudWatch Application Signals に対する新しい読み取り専用アクセス許可を追加しました。 このポリシーを持つユーザーが自然言語プロンプトから CloudWatch Metrics Insights クエリ文字列を生成できるように、`cloudwatch:GenerateQuery` アクセス許可を追加しました。	2023 年 12 月 5 日

変更	説明	日付
CloudWatchApplicationSignalsServiceRolePolicy — 新しいポリシー	CloudWatch に新しいポリシー CloudWatchApplicationSignalsServiceRolePolicy が追加されました。 CloudWatchApplicationSignalsServiceRolePolicy により、CloudWatch Logs データ、X-Ray トレースデータ、CloudWatch メトリクスデータ、タグ付けデータを収集するための、今後の機能へのアクセス許可が付与されます。	2023 年 11 月 9 日
AWSServiceRoleForCloudWatchMetrics_DbPerfInsightsServiceRolePolicy – 新しいポリシー	CloudWatch に新しいポリシー AWSServiceRoleForCloudWatchMetrics_DbPerfInsightsServiceRolePolicy が追加されました。 AWSServiceRoleForCloudWatchMetrics_DbPerfInsightsServiceRolePolicy は、ユーザーに代わってデータベースから Performance Insights メトリクスを取得するアクセス許可を CloudWatch に付与します。	2023 年 9 月 20 日

変更	説明	日付
CloudWatchReadOnlyAccess – 既存のポリシーに対する更新	CloudWatch が、CloudWatchReadOnlyAccess に対するアクセス許可を追加しました。 このポリシーを持つユーザーが Application Auto Scaling ポリシーに関する情報にアクセスできるように `application-autoscaling:DescribeScalingPolicies` アクセス許可が追加されました。	2023 年 9 月 14 日
CloudWatchFullAccessV2 — 新しいポリシー	CloudWatch に新しいポリシー CloudWatchFullAccessV2 が追加されました。 CloudWatchFullAccessV2 は、CloudWatch アクションとリソースへのフルアクセスを許可すると同時に、Amazon SNS や Amazon EC2 Auto Scaling などの他のサービスに付与されるアクセス許可の範囲をより適切に設定します。詳細については、「CloudWatchFullAccess V2」を参照してください。	2023 年 8 月 1 日

変更	説明	日付
AWSServiceRoleForInternetMonitor – 既存のポリシーへの更新	Amazon CloudWatch Internet Monitor が、Network Load Balancer リソースをモニタリングするための新しいアクセス許可を追加しました。 Internet Monitor が NLB リソースのフローログを分析して顧客の Network Load Balancer トラフィックを監視できるようにするには、`elasticloadbalancing:DescribeLoadBalancers` および `ec2:DescribeNetworkInterfaces` アクセス許可が必要です。 詳細については、「Amazon CloudWatch Internet Monitor の使用」を参照してください。	2023 年 7 月 15 日

変更	説明	日付
CloudWatchReadOnlyAccess – 既存のポリシーに対する更新	CloudWatch が、CloudWatchReadOnlyAccess に対する許可を追加しました。 `logs:StartLiveTail` および `logs:StopLiveTail` 許可が追加されたため、このポリシーを持つユーザーがコンソールを使用して CloudWatch Logs ライブテールセッションを開始および停止できるようになりました。詳細については、「ライブテールを使用してほぼリアルタイムでログを表示する」を参照してください。	2023 年 6 月 6 日
CloudWatchCrossAccountSharingConfiguration – 新しいポリシー	CloudWatch が、CloudWatch メトリクスを共有する CloudWatch クロスアカウントオブザーバビリティのリンクを管理できるようにする新しいポリシーを追加しました。 詳細については、「CloudWatch のクロスアカウントオブザーバビリティ」を参照してください。	2022 年 11 月 27 日

変更	説明	日付
OAMFullAccess – 新しいポリシー	CloudWatch が、CloudWatch クロスアカウントオブザーバビリティのリンクとシンクを完全に管理できるようにする新しいポリシーを追加しました。 詳細については、「CloudWatch のクロスアカウントオブザーバビリティ」を参照してください。	2022 年 11 月 27 日
OAMReadOnlyAccess – 新しいポリシー	CloudWatch が、CloudWatch クロスアカウントオブザーバビリティのリンクとシンクに関する情報を表示できるようにする新しいポリシーを追加しました。 詳細については、「CloudWatch のクロスアカウントオブザーバビリティ」を参照してください。	2022 年 11 月 27 日

変更	説明	日付
CloudWatchFullAccess – 既存のポリシーに対する更新	CloudWatch が、CloudWatchFullAccess に対する許可を追加しました。 このポリシーを持つユーザーがコンソールを使用して、CloudWatch のクロスアカウントオブザーバビリティでソースアカウントから共有されたデータを表示できるようにするための `oam:ListSinks` および `oam:ListAttachedLinks` 許可が追加されました。	2022 年 11 月 27 日
CloudWatchReadOnlyAccess – 既存のポリシーに対する更新	CloudWatch が、CloudWatchReadOnlyAccess に対する許可を追加しました。 このポリシーを持つユーザーがコンソールを使用して、CloudWatch のクロスアカウントオブザーバビリティでソースアカウントから共有されたデータを表示できるようにするための `oam:ListSinks` および `oam:ListAttachedLinks` 許可が追加されました。	2022 年 11 月 27 日

変更	説明	日付
AmazonCloudWatchRUMServiceRolePolicy – 既存のポリシーへの更新	CloudWatch RUM が AmazonCloudWatchRUMServiceRolePolicy の条件キーを更新しました。 CloudWatch RUM がカスタムメトリクスをカスタムメトリクス名前空間に送信できるように、`"Condition": { "StringEquals": { "cloudwatch:namespace": "AWS/RUM" } }` 条件キーが次のように変更されました。 `"Condition": { "StringLike": { "cloudwatch:namespace": [ "RUM/CustomMetrics/*", "AWS/RUM" ] } }`	2023 年 2 月 2 日

変更	説明	日付
AmazonCloudWatchRUMReadOnlyAccess – ポリシーを更新	CloudWatch で AmazonCloudWatchRUMReadOnlyAccess ポリシーにアクセス許可が追加されました。 CloudWatch RUM が拡張メトリクスを CloudWatch と Evidently に送信できるように、`rum:ListRumMetricsDestinations` および `rum:BatchGetRumMetricsDefinitions` アクセス許可が追加されました。	2022 年 10 月 27 日
AmazonCloudWatchRUMServiceRolePolicy – 既存のポリシーへの更新	CloudWatch RUM で AmazonCloudWatchRUMServiceRolePolicy へのアクセス許可が追加されました。 CloudWatch RUM が拡張メトリクスを CloudWatch に送信できるように、`cloudwatch:PutMetricData` アクセス許可が追加されました。	2022 年 10 月 26 日

変更	説明	日付
CloudWatchEvidentlyReadOnlyAccess — 既存のポリシーへの更新	CloudWatch Evidently が、CloudWatchEvidentlyReadOnlyAccess にアクセス許可を追加しました。 このポリシーを持つユーザーが、作成された Evidently のオーディエンスセグメントを表示できるように `evidently:GetSegment` 、`evidently:ListSegments` 、および `evidently:ListSegmentReferences` のアクセス許可が追加されました。	2022 年 8 月 12 日
CloudWatchSyntheticsFullAccess – 既存のポリシーを更新	CloudWatch Synthetics が、CloudWatchSyntheticsFullAccess にアクセス許可を追加しました。 Canary が削除されたときに CloudWatch Synthetics が関連リソースを削除できるように、`lambda:DeleteFunction` および `lambda:DeleteLayerVersion` アクセス許可が追加されました。Canary の IAM ロールにアタッチされているポリシーを顧客が表示できるように `iam:ListAttachedRolePolicies` が追加されました。	2022 年 5 月 6 日

変更	説明	日付
AmazonCloudWatchRUMFullAccess – 新しいポリシー	CloudWatch に CloudWatch RUM の完全な管理を有効にする新しいポリシーが追加されました。 CloudWatch RUM を使用すると、ウェブアプリケーションの実際のユーザーモニターリングを実行できます。詳細については、「CloudWatch RUM を使用する」を参照してください。	2021 年 11 月 29 日
AmazonCloudWatchRUMReadOnlyAccess – 新しいポリシー	CloudWatch に CloudWatch RUM への読み取り専用のアクセス権限を有効にする新しいポリシーが追加されました。 CloudWatch RUM を使用すると、ウェブアプリケーションの実際のユーザーモニターリングを実行できます。詳細については、「CloudWatch RUM を使用する」を参照してください。	2021 年 11 月 29 日

変更	説明	日付
CloudWatchEvidentlyFullAccess – 新しいポリシー	CloudWatch に CloudWatch Evidently の完全な管理を有効にする新しいポリシーが追加されました。 CloudWatch Evidently では、ウェブアプリケーションの A/B 実験を実行し、徐々にロールアウトすることができます。詳細については、「CloudWatch Evidently での起動と A/B 実験を実行する」を参照してください。	2021 年 11 月 29 日
CloudWatchEvidentlyReadOnlyAccess – 新しいポリシー	CloudWatch に CloudWatch Evidently への読み取り専用のアクセスを有効にする新しいポリシーが追加されました。 CloudWatch Evidently では、ウェブアプリケーションの A/B 実験を実行し、徐々にロールアウトすることができます。詳細については、「CloudWatch Evidently での起動と A/B 実験を実行する」を参照してください。	2021 年 11 月 29 日
AWSServiceRoleForCloudWatchRUM – 新しい管理ポリシー	CloudWatch にサービスにリンクされた新しいロールのポリシーが追加されて、CloudWatch RUM がモニターリングデータを他の関連する AWS のサービスに公開できるようになりました。	2021 年 11 月 29 日

変更	説明	日付
CloudWatchSyntheticsFullAccess – 既存のポリシーを更新	CloudWatch Synthetics で CloudWatchSyntheticsFullAccess へのアクセス許可が追加され、また、1 つのアクセス許可の範囲も変更されました。 `kms:ListAliases` アクセス許可が追加され、canary アーティファクトの暗号化に使用できる AWS KMS キーのリストをユーザーが表示できるようになりました。`kms:DescribeKey` アクセス許可が追加され、canary アーティファクトの暗号化に使用できるキーの詳細をユーザーが表示できるようになりました。そして、ユーザーが canary アーティファクトを復号化できるようにする `kms:Decrypt` アクセス許可が追加されました。この復号化機能は、Amazon S3 バケット内のリソースでの使用に制限されています。 `s3:GetBucketLocation` アクセス許可の `Resource` 範囲が `*` から `arn:aws:s3:::*` に変更されました。	2021 年 9 月 29 日

変更	説明	日付
CloudWatchSyntheticsFullAccess – 既存のポリシーを更新	CloudWatch Synthetics が、CloudWatchSyntheticsFullAccess に許可を追加しました。 このポリシーを持つユーザーが Canary のランタイムバージョンを変更できるように、`lambda:UpdateFunctionCode` 許可が追加されました。	2021 年 7 月 20 日
AWSCloudWatchAlarms_ActionSSMIncidentsServiceRolePolicy – 新しい管理ポリシー	CloudWatch は、AWS Systems Manager Incident Manager でインシデントを作成できるように、新しい IAM マネージドポリシーを追加しました。	2021 年 5 月 10 日
CloudWatchAutomaticDashboardsAccess – 既存のポリシーに対する更新	CloudWatch が、CloudWatchAutomaticDashboardsAccess 管理ポリシーにアクセス権限を追加しました。クロスアカウントダッシュボードユーザーが、CloudWatch Synthetics Canary 実行の詳細を表示するための `synthetics:DescribeCanariesLastRun` アクセス許可が、このポリシーに追加されました。	2021 年 4 月 20 日
CloudWatch が変更の追跡を開始しました	CloudWatch が、AWS マネージドポリシーの変更の追跡を開始しました。	2021 年 4 月 14 日

条件キーを使用した CloudWatch 名前空間へのアクセスの制限

IAM 条件キーを使用して、ユーザーがメトリクスを公開する先を、指定した CloudWatch 名前空間に限定します。

1 つの名前空間でのみ公開を許可する

次のポリシーでは、ユーザーがメトリクスを公開する先を、MyCustomNamespace という名前空間に限定します。

```
{
    "Version": "2012-10-17",
    "Statement": {
        "Effect": "Allow",
        "Resource": "*",
        "Action": "cloudwatch:PutMetricData",
        "Condition": {
            "StringEquals": {
                "cloudwatch:namespace": "MyCustomNamespace"
            }
        }
    }
}
```

名前空間から公開を除外する

次のポリシーでは、ユーザーがメトリクスを公開する先として、CustomNamespace2 以外のすべての名前空間を許可します。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Resource": "*",
            "Action": "cloudwatch:PutMetricData"
        },
        {
            "Effect": "Deny",
            "Resource": "*",
            "Action": "cloudwatch:PutMetricData",
            "Condition": {
```

```
                "StringEquals": {
                    "cloudwatch:namespace": "CustomNamespace2"
                }
            }
        }
    ]
}
```

Contributor Insights のユーザーのロググループへのアクセスを制限するための条件キーの使用

Contributor Insights でルールを作成し、その結果を表示するには、ユーザーに `cloudwatch:PutInsightRule` アクセス許可が必要です。デフォルトでは、このアクセス許可を持つユーザーは、CloudWatch Logs のロググループを評価する Contributor Insights ルールを作成して、結果を確認できます。結果には、これらのロググループのコントリビューターデータを含めることができます。

条件キーを持つ IAM ポリシーを作成して、一部のロググループに対して Contributor Insights ルールを作成するアクセス許可をユーザーに付与し、他のロググループからこのデータを表示しないようにすることができます。

IAM ポリシーの `Condition` 要素の詳細については、「IAM JSON ポリシーの要素: 条件」を参照してください。

特定のロググループのみについて、ルールの書き込みおよび結果の表示へのアクセスを許可する

次のポリシーでは、`AllowedLogGroup` という名前のロググループと `AllowedWildCard` で始まる名前を持つすべてのロググループのルールを作成し、結果を表示するためのユーザーアクセスを許可します。他のロググループには、書き込みルールへのアクセスやルールの結果の表示する権限は付与されません。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowCertainLogGroups",
            "Effect": "Allow",
            "Action": "cloudwatch:PutInsightRule",
            "Resource": "arn:aws:cloudwatch:*:*:insight-rule/*",
            "Condition": {
                "ForAllValues:StringEqualsIgnoreCase": {
```

```
                    "cloudwatch:requestInsightRuleLogGroups": [
                        "AllowedLogGroup",
                        "AllowedWildcard*"
                    ]
                }
            }
        }
    ]
}
```

特定のロググループに対するルールの書き込みを拒否するが、他のすべてのロググループに対するルールの書き込みを許可する

次のポリシーでは、ExplicitlyDeniedLogGroup という名前のロググループのルールの書き込みおよびルールの結果の表示に対するユーザーアクセスを明示的に拒否しますが、他のすべてのロググループのルールの書き込みおよびルールの結果の表示は許可します。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowInsightRulesOnLogGroupsByDefault",
            "Effect": "Allow",
            "Action": "cloudwatch:PutInsightRule",
            "Resource": "arn:aws:cloudwatch:*:*:insight-rule/*"

        },
        {
            "Sid": "ExplicitDenySomeLogGroups",
            "Effect": "Deny",
            "Action": "cloudwatch:PutInsightRule",
            "Resource": "arn:aws:cloudwatch:*:*:insight-rule/*",
            "Condition": {
                "ForAllValues:StringEqualsIgnoreCase": {
                    "cloudwatch:requestInsightRuleLogGroups": [
                        "/test/alpine/ExplicitlyDeniedLogGroup"
                    ]
                }
            }
        }
    ]
}
```

アラームアクションを制限するための条件キーの使用

CloudWatch アラームの状態が変化すると、EC2 インスタンスの停止と終了、Systems Manager アクションの実行など、さまざまなアクションを実行できます。これらのアクションは、アラームが ALARM、OK、INSUFICIENT_DATA などの任意の状態に変更されたときに開始できます。

`cloudwatch:AlarmActions` 条件キーを使用して、アラームの状態が変化したときに指定したアクションを実行することしかできないアラームをユーザーが作成できるようにします。例えば、EC2 アクションではないアクションのみを実行できるアラームの作成をユーザーに許可できます。

Amazon SNS 通知を送信したり、Systems Manager のアクションを実行したりできるアラームの作成をユーザーに許可する

次のポリシーでは、ユーザーが作成できるのが、Amazon SNS 通知を送信し、Systems Manager のアクションを実行することしかできないアラームであるように制限します。ユーザーは EC2 アクションを実行するアラームを作成できません。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "CreateAlarmsThatCanPerformOnlySNSandSSMActions",
            "Effect": "Allow",
            "Action": "cloudwatch:PutMetricAlarm",
            "Resource": "*",
            "Condition": {
                "ForAllValues:StringLike": {
                    "cloudwatch:AlarmActions": [
                        "arn:aws:sns:*",
                        "arn:aws:ssm:*"
                    ]
                }
            }
        }
    ]
}
```

CloudWatch のサービスにリンクされたロールの使用

Amazon CloudWatch は AWS Identity and Access Management (IAM) サービスにリンクされたロールを使用します。サービスにリンクされたロールは、CloudWatch に直接リンクされた一意のタイ

プの IAM ロールです。サービスにリンクされたロールは、CloudWatch によって事前定義されており、お客様の代わりにサービスから他の AWS のサービスを呼び出す必要のある許可がすべて含まれています。

CloudWatch のサービスにリンクされたロールを使用すると、必要なアクセス権限を手動で追加しなくても、Amazon EC2 インスタンスを終了、停止、または再起動できる CloudWatch アラームを設定できます。別のサービスにリンクされたロールを使用すると、モニターリングアカウントが、指定した他のアカウントの CloudWatch データにアクセスし、クロスアカウントクロスリージョンダッシュボードを構築できるようになります。

CloudWatch は、サービスにリンクされたロールのアクセス権限を定義します。特に定義されている場合を除き、CloudWatch はそのロールのみを引き受けることができます。定義される許可は、信頼ポリシーと許可ポリシーに含まれており、その許可ポリシーを他の IAM エンティティにアタッチすることはできません。

ロールを削除するには、まず関連リソースを削除します。この制限により、不注意でリソースにアクセスするアクセス許可の削除が防止され、CloudWatch リソースは保護されます。

サービスにリンクされたロールをサポートする他のサービスについては、「IAM と連携するAWS サービス」を参照して、サービスにリンクされたロール列がはいになっているサービスを見つけてください。サービスにリンクされたロールに関するドキュメントをサービスで表示するには、[はい] リンクを選択します。

CloudWatch アラーム EC2 アクションについてのサービスにリンクされたロールの許可

CloudWatch は、AWSServiceRoleForCloudWatchEvents という名前のサービスにリンクされたロールを使用します – CloudWatch は、このサービスにリンクされたロールを使用して Amazon EC2 アラームアクションを実行します。

AWSServiceRoleforCloudWatchEvents サービスにリンクされたロールは、CloudWatch Events サービスを信頼してロールを継承します。CloudWatch Events は、アラームにより呼び出されたときにインスタンスアクションを終了、停止、または再起動を呼び出します。

AWSServiceRoleForCloudWatchEvents サービスにリンクされたロールのアクセス許可ポリシーでは、CloudWatch Events が Amazon EC2 インスタンスで以下のアクションを実行できます。

- `ec2:StopInstances`
- `ec2:TerminateInstances`
- `ec2:RecoverInstances`

- ec2:DescribeInstanceRecoveryAttribute
- ec2:DescribeInstances
- ec2:DescribeInstanceStatus

AWSServiceRoleForCloudWatchCrossAccount サービスにリンクされたロールのアクセス権限ポリシーでは、CloudWatch は、次のアクションを完了することができます。

- sts:AssumeRole

CloudWatch Application Signals のサービスリンクロールのアクセス許可

CloudWatch Application Signals では、AWSServiceRoleForCloudWatchApplicationSignals という名前のサービスリンクロールを使用します。CloudWatch は、このサービスリンクロールを使用して、CloudWatch Application Signals を利用できるアプリケーションから CloudWatch ログデータ、X-Ray トレースデータ、CloudWatch メトリクスデータ、タグデータを収集します。

AWSServiceRoleForCloudWatchApplicationSignals サービスリンクロールは、ロールを継承するために CloudWatch Application Signals サービスを信頼します。Application Signals は、ユーザーのアカウントからログ、トレース、メトリクス、タグのデータを収集します。

AWSServiceRoleForCloudWatchApplicationSignals には IAM ポリシーがアタッチされており、このポリシーは CloudWatchApplicationSignalsServiceRolePolicy という名前です。このポリシーは、CloudWatch Application Signals が他の関連 AWS サービスからモニタリングデータとタグデータを収集できるように必要なアクセス許可を付与します。これには、Application Signals が次のアクションを完了するために必要なアクセス許可が含まれています。

- xray:GetServiceGraph
- logs:StartQuery
- logs:GetQueryResults
- cloudwatch:GetMetricData
- cloudwatch:ListMetrics
- tag:GetResources

CloudWatchApplicationSignalsServiceRolePolicy ポリシーの完全な内容は次のとおりです。

```
{
```

```
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "XRayPermission",
            "Effect": "Allow",
            "Action": [
                "xray:GetServiceGraph"
            ],
            "Resource": [
                "*"
            ],
            "Condition": {
                "StringEquals": {
                    "aws:ResourceAccount": "${aws:PrincipalAccount}"
                }
            }
        },
        {
            "Sid": "CWLogsPermission",
            "Effect": "Allow",
            "Action": [
                "logs:StartQuery",
                "logs:GetQueryResults"
            ],
            "Resource": [
                "arn:aws:logs:*:*:log-group:/aws/appsignals/*:*",
                "arn:aws:logs:*:*:log-group:/aws/application-signals/data:*"
            ],
            "Condition": {
                "StringEquals": {
                    "aws:ResourceAccount": "${aws:PrincipalAccount}"
                }
            }
        },
        {
            "Sid": "CWListMetricsPermission",
            "Effect": "Allow",
            "Action": [
                "cloudwatch:ListMetrics"
            ],
            "Resource": [
                "*"
            ],
            "Condition": {
```

```
                "StringEquals": {
                    "aws:ResourceAccount": "${aws:PrincipalAccount}"
                }
            }
        },
        {
            "Sid": "CWGetMetricDataPermission",
            "Effect": "Allow",
            "Action": [
                "cloudwatch:GetMetricData"
            ],
            "Resource": [
                "*"
            ]
        },
        {
            "Sid": "TagsPermission",
            "Effect": "Allow",
            "Action": [
                "tag:GetResources"
            ],
            "Resource": [
                "*"
            ],
            "Condition": {
                "StringEquals": {
                    "aws:ResourceAccount": "${aws:PrincipalAccount}"
                }
            }
        }
    ]
}
```

CloudWatch アラーム Systems Manager OpsCenter アクションに対するサービスにリンクされたロールの許可

CloudWatch は、AWSServiceRoleForCloudWatchAlarms_ActionSSM という名前のサービスにリンクされたロールを使用します – CloudWatch は、CloudWatch アラームが ALARM 状態になったときに Systems Manager OpsCenter アクションを実行するために、このサービスにリンクされたロールを使用します。

AWSServiceRoleForCloudWatchAlarms_ActionSSM サービスにリンクされたロールは、CloudWatch サービスを信頼してロールを継承します。CloudWatch アラームは、アラームによって呼び出されると、Systems Manager OpsCenter アクションを呼び出します。

AWSServiceRoleForCloudWatchAlarms_ActionSSM サービスにリンクされたロールのアクセス許可ポリシーでは、Systems Manager は、次のアクションを実行できます。

- `ssm:CreateOpsItem`

CloudWatch アラーム Systems Manager Incident Manager アクションのサービスにリンクされたロールの許可

CloudWatch は、AWSServiceRoleForCloudWatchAlarms_ActionSSMIncidents という名前のサービスにリンクされたロールを使用します – CloudWatch アラームが ALARM 状態になると、CloudWatch はこのサービスにリンクされたロールを使用して Incident Manager インシデントを開始します。

AWSServiceRoleForCloudWatchAlarms_ActionSSMIncidents サービスにリンクされたロールは、CloudWatch サービスを信頼してロールを継承します。CloudWatch アラームは、アラームによって呼び出されると、Systems Manager Incident Manager アクションを呼び出します。

AWSServiceRoleForCloudWatchAlarms_ActionSSMIncidents サービスにリンクされたロールのアクセス許可ポリシーでは、Systems Manager は、次のアクションを実行できます。

- `ssm-incidents:StartIncident`

CloudWatch クロスアカウントクロスリージョンに対するサービスにリンクされたロールの許可

CloudWatch では、AWSServiceRoleForCloudWatchCrossAccount と呼ばれるサービスにリンクされたロールを使用します – CloudWatch は、このロールを使用し、指定した他の AWS アカウントの CloudWatch データにアクセスします。SLR は、CloudWatch サービスが共有アカウントのロールを継承できるようにするため、継承ロールアクセス権限のみを提供します。これは、データへのアクセスを提供する共有ロールです。

AWSServiceRoleForCloudWatchCrossAccount サービスにリンクされたロールのアクセス権限ポリシーでは、CloudWatch は、次のアクションを完了することができます。

- `sts:AssumeRole`

AWSServiceRoleForCloudWatchCrossAccount サービスにリンクされたロールは、ロールを継承するために CloudWatch サービスを信頼します。

CloudWatch データベース Performance Insights に対するサービスリンクロールのアクセス許可

CloudWatch Logs は AWSServiceRoleForLogDelivery という名前のサービスにリンクされたロールを使用します。— CloudWatch は、このロールを使用して、アラーム作成とスナップショット作成のための Performance Insights メトリクスを取得します。

サービスにリンクされたロール AAWSServiceRoleForCloudWatchMetrics_DbPerfInsights には、AWSServiceRoleForCloudWatchMetrics_DbPerfInsightsServiceRolePolicy IAM ポリシーがアタッチされています。このポリシーの内容は次のとおりです。

```
{
 "Version": "2012-10-17",
 "Statement": [
  {
   "Effect": "Allow",
   "Action": [
    "pi:GetResourceMetrics"
   ],
   "Resource": "*",
   "Condition": {
    "StringEquals": {
     "aws:ResourceAccount": "${aws:PrincipalAccount}"
    }
   }
  }
 ]
}
```

サービスにリンクされたロール AWSServiceRoleForCloudWatchCrossAccount は、CloudWatch サービスを信頼してロールを引き受けます。

CloudWatch のサービスにリンクされたロールの作成

これらのサービスにリンクされたどのロールも手動で作成する必要はありません。AWS Management Console、IAM CLI、または IAM API でアラームを初めて作成すると、CloudWatch によって AWSServiceRoleForCloudWatchEvents と AWSServiceRoleForCloudWatchAlarms_ActionSSM が自動的に作成されます。

初めてサービスとトポロジを検出できるようにした場合、Application Signals によって AWSServiceRoleForCloudWatchApplicationSignals が自動的に作成されます。

アカウントをクロスアカウントクロスリージョン機能のモニターリングアカウントとして初めて有効にすると、CloudWatch によって AWSServiceRoleForCloudWatchCrossAccount が自動的に作成されます。

DB_PERF_INSIGHTS Metric Math 関数を使用するアラームを初めて作成すると、CloudWatch は AWSServiceRoleForCloudWatchMetrics_DbPerfInsights を自動的に作成します。

詳細については、IAM ユーザーガイドの「サービスにリンクされたロールの作成」を参照してください。

CloudWatch のサービスにリンクされたロールの編集

CloudWatch では、AWSServiceRoleForCloudWatchEvents、AWSServiceRoleForCloudWatchAlarms_ActionSSM、AWSSe または AWSServiceRoleForCloudWatchMetrics_DbPerfInsights ロールを編集できません。これらのロールは多くのエンティティにより参照されるため、ロールを作成した後その名前を変更することはできません。ただし、IAM を使用したロールの説明の編集はできます。

サービスにリンクされたロールの説明の編集 (IAM コンソール)

サービスにリンクされたロールの説明は、IAM コンソールを使用して編集できます。

サービスにリンクされたロールの説明を編集するには (コンソール)

1. IAM コンソールのナビゲーションペインで [ロール] を選択します。
2. 変更するロールの名前を選択します。
3. [Role description] の右端にある [Edit] を選択します。
4. ボックスに新しい説明を入力し、[Save] を選択します。

サービスにリンクされたロールの説明の編集 (AWS CLI)

AWS Command Line Interface から IAM コマンドを使用して、サービスにリンクされたロールの説明を編集できます。

サービスにリンクされたロールの説明を変更するには (AWS CLI)

1. (オプション) ロールの現在の説明を表示するには、次のコマンドを使用します。

```
$ aws iam get-role --role-name role-name
```

AWS CLI コマンドでは、ARN ではなくロール名を使用してロールを参照します。例えば、ロールの ARN が `arn:aws:iam::123456789012:role/myrole` である場合、そのロールを **myrole** と参照します。

2. サービスにリンクされたロールの説明を更新するには、次のコマンドを使用します。

```
$ aws iam update-role-description --role-name role-name --description description
```

サービスにリンクされたロールの説明の編集 (IAM API)

サービスにリンクされたロールの説明は、IAM API を使用して編集できます。

サービスにリンクされたロールの説明を変更するには (API)

1. (オプション) ロールの現在の説明を表示するには、次のコマンドを使用します。

   GetRole
2. ロールの説明を更新するには、次のコマンドを使用します。

   UpdateRoleDescription

CloudWatch のサービスにリンクされたロールの削除

EC2 インスタンスを自動的に停止、終了、または再起動するアラームがなくなった場合、AWSServiceRoleForCloudWatchEvents ロールを削除することをお勧めします。

Systems Manager OpsCenter アクションを実行するアラームがなくなった場合は、AWSServiceRoleForCloudWatchAlarms_ActionSSM ロールを削除することをお勧めします。

`DB_PERF_INSIGHTS` Metric Math 関数を使用するアラームをすべて削除する場合は、サービスにリンクされたロール AWSServiceRoleForCloudWatchMetrics_DBPerfInsights を削除することをお勧めします。

そうすることで、使用していないエンティティがアクティブにモニターリングされたり、メンテナンスされたりすることがなくなります。ただし、削除する前に、サービスにリンクされた役割をクリーンアップする必要があります。

サービスにリンクされたロールのクリーンアップ

IAM を使用してサービスにリンクされたロールを削除するには、まずそのロールにアクティブなセッションがないことを確認し、そのロールで使用されているリソースをすべて削除する必要があります。

サービスにリンクされたロールにアクティブなセッションがあるかどうかを、IAM コンソールで確認するには

1. IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
2. ナビゲーションペインで [Roles (ロール)] を選択します。AWSServiceRoleForCloudWatchEvents ロールの名前 (チェックボックスではない) を選択します。
3. 選択したロールの [概要] ページで [アクセスアドバイザー] を選択し、サービスにリンクされたロールの最新のアクティビティを確認します。

> **Note**
>
> CloudWatch が AWSServiceRoleForCloudWatchEvents ロールを使用しているかどうか不明な場合は、ロールを削除してみてください。サービスでロールが使用されている場合、削除は失敗し、ロールが使用されている リージョンが表示されます。ロールが使用されている場合は、ロールを削除する前にセッションが終了するのを待つ必要があります。サービスにリンクされたロールのセッションを取り消すことはできません。

サービスにリンクされたロールの削除 (IAM コンソール)

IAM コンソールを使用して、サービスにリンクされたロールを削除できます。

サービスにリンクされたロールを削除するには (コンソール)

1. IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
2. ナビゲーションペインで [Roles (ロール)] を選択します。ロール名または行そのものではなく、削除するロールの名前の横にあるチェックボックスをオンにします。
3. [ロールのアクション] で、[ロールの削除] を選択します。
4. 確認ダイアログボックスで、サービスの最終アクセス時間データを確認します。これは、選択したそれぞれのロールの AWS サービスへの最終アクセス時間を示します。これは、そのロールが

現在アクティブであるかどうかを確認するのに役立ちます。続行するには、[はい、削除します] を選択します。

5. IAM コンソール通知を見て、サービスにリンクされたロールの削除の進行状況を監視します。IAM サービスにリンクされたロールの削除は非同期であるため、削除するロールを送信すると、削除タスクは成功または失敗する可能性があります。タスクが失敗した場合は、通知から [詳細を表示] または [リソースを表示] を選択して、削除が失敗した理由を知ることができます。そのロールで使用中のリソースがサービスにあるために削除に失敗した場合、この失敗の理由にはリソースのリストも含まれます。

サービスにリンクされたロールの削除 (AWS CLI)

AWS Command Line Interface から IAM コマンドを使用して、サービスにリンクされたロールを削除できます。

サービスにリンクされたロールを削除するには (AWS CLI)

1. サービスにリンクされているロールは、使用されている、または関連するリソースがある場合は削除できないため、削除リクエストを送信する必要があります。これらの条件が満たされない場合、そのリクエストは拒否される可能性があります。レスポンスから `deletion-task-id` を取得して、削除タスクのステータスを確認する必要があります。サービスにリンクされたロールの削除リクエストを送信するには、次のコマンドを入力します。

```
$ aws iam delete-service-linked-role --role-name service-linked-role-name
```

2. 削除タスクのステータスを確認するには、次のコマンドを入力します。

```
$ aws iam get-service-linked-role-deletion-status --deletion-task-id deletion-task-id
```

削除タスクのステータスは、`NOT_STARTED`、`IN_PROGRESS`、`SUCCEEDED`、または `FAILED` となります。削除が失敗した場合は、失敗した理由がコールによって返され、トラブルシューティングが可能になります。

サービスにリンクされたロールの削除 (IAM API)

IAM API を使用して、サービスにリンクされたロールを削除できます。

サービスにリンクされたロールを削除するには (API)

1. サービスにリンクされたロールの削除リクエストを送信するには、DeleteServiceLinkedRole を呼び出します。リクエストで、削除するロール名を指定します。

   サービスにリンクされているロールは、使用されている、または関連するリソースがある場合は削除できないため、削除リクエストを送信する必要があります。これらの条件が満たされない場合、そのリクエストは拒否される可能性があります。レスポンスから DeletionTaskId を取得して、削除タスクのステータスを確認する必要があります。
2. 削除タスクのステータスを確認するには、GetServiceLinkedRoleDeletionStatus を呼び出します。リクエストで DeletionTaskId を指定します。

   削除タスクのステータスは、NOT_STARTED、IN_PROGRESS、SUCCEEDED、または FAILED となります。削除が失敗した場合は、失敗した理由がコールによって返され、トラブルシューティングが可能になります。

AWS のサービスにリンクされたロールへの CloudWatch の更新

CloudWatch の AWS マネージドポリシーの更新に関する詳細を、このサービスがこれらの変更の追跡を開始した以降の分について表示します。このページの変更に関する自動通知を入手するには、CloudWatch ドキュメントの履歴ページから、RSS フィードにサブスクライブしてください。

変更	説明	日付
AWSServiceRoleForCloudWatchApplicationSignals – サービスにリンクされたロールポリシーのアクセス許可の更新	CloudWatch は、このロールによって付与される logs:StartQuery および logs:GetQueryResults のアクセス許可のスコープにさらにロググループを追加します。	2024 年 4 月 24 日
AWSServiceRoleForCloudWatchApplicationSignals - 新しいサービスリンクロール	CloudWatch Application Signals を利用できるアプリケーションから CloudWatch	2023 年 11 月 9 日

変更	説明	日付
	ログデータ、X-Ray トレースデータ、CloudWatch メトリクスデータ、タグデータを収集できるように、この新しいサービスリンクロールを追加しました。	
AWSServiceRoleForCloudWatchMetrics_DbPerfInsights – サービスにリンクされた新しいロール	このサービスリンクロールが追加されたことで、CloudWatch がアラームやスナップショット用の Performance Insights メトリクスを取得できるようにしました。このロールには IAM ポリシーがアタッチされています。このポリシーにより、ユーザーに代わって Performance Insights メトリクスを取得するアクセス許可が CloudWatch に付与されます。	2023 年 9 月 13 日
AWSServiceRoleForCloudWatchAlarms_ActionSSMIncidents – サービスにリンクされた新しいロール	CloudWatch は、AWS Systems Manager Incident Manager でインシデントを作成できるように、サービスにリンクされた新しいロールを追加しました。	2021 年 4 月 26 日
CloudWatch が変更の追跡を開始しました	CloudWatch は、サービスにリンクされたロールの変更の追跡を開始しました。	2021 年 4 月 26 日

CloudWatch RUM のサービスにリンクされたロールの使用

CloudWatch RUM は AWS Identity and Access Management (IAM) サービスにリンクされたロールを使用します。サービスにリンクされたロールは、RUM に直接リンクされた一意のタイプの IAM ロールです。サービスリンクロールは RUM によって事前に定義されており、サービスがユーザーに代わって他の AWS サービスを呼び出すために必要なアクセス許可がすべて含まれています。

RUM は、サービスにリンクされたロールのアクセス許可を定義します。特に定義されている場合を除き、RUM のみがそのロールを引き受けることができます。定義したアクセス許可には、信頼ポリシーと許可ポリシーが含まれます。この許可ポリシーを他の IAM エンティティにアタッチすることはできません。

ロールを削除するには、まずそのロールの関連リソースを削除します。この制限により、リソースにアクセスするアクセス許可の不注意による削除が防止され、RUM リソースは保護されます。

サービスリンクロールをサポートする他のサービスについては、「IAM と連動する AWS のサービス」を参照し、[Service-linked role (サービスリンクロール)] の列内で [Yes (はい)] と表記されたサービスを確認してください。サービスにリンクされたロールに関するドキュメントをサービスで表示するには、[Yes] (はい) リンクを選択します。

RUM のサービスにリンクされたロールのアクセス許可

RUM では、サービスにリンクされたロールとして AWSServiceRoleForCloudWatchRUM を使用します – このロールは、X-Ray トレースを有効にするアプリケーションモニター用に、RUM がアカウントに AWS X-Ray トレースデータを送信できるようにします。

サービスにリンクされた AWSServiceRoleForCloudWatchRUM ロールは、X-Ray サービスを信頼してロールを引き受けます。X-Ray はトレースデータをアカウントに送信します。

サービスにリンクされた AWSServiceRoleForCloudWatchRUM ロールには、AmazonCloudWatchRUMServiceRolePolicy という IAM ポリシーがアタッチされています。このポリシーは、モニタリングデータを他の関連する AWS サービスに公開するアクセス許可を CloudWatch RUM に付与します。これには、RUM が次のアクションを完了できるようにするアクセス許可が含まれています。

- `xray:PutTraceSegments`
- `cloudwatch:PutMetricData`

AmazonCloudWatchRUMServiceRolePolicy の詳細な内容は次のとおりです。

```
{
 "Version": "2012-10-17",
 "Statement": [
  {
   "Effect": "Allow",
   "Action": [
    "xray:PutTraceSegments"
   ],
   "Resource": [
    "*"
   ]
  },
  {
   "Effect": "Allow",
   "Action": "cloudwatch:PutMetricData",
   "Resource": "*",
   "Condition": {
    "StringLike": {
     "cloudwatch:namespace": [
      "RUM/CustomMetrics/*",
      "AWS/RUM"
     ]
    }
   }
  }
 ]
}
```

RUM のサービスにリンクされたロールの作成

サービスにリンクされた CloudWatch RUM のロールを手動で作成する必要はありません。X-Ray トレーシングを有効にしたアプリケーションモニターを初めて作成するとき、または X-Ray トレースを使用するようにアプリケーションモニターを更新するとき、RUM によって AWSServiceRoleForCloudWatchRUM が作成されます。

詳細については、「IAM ユーザーガイド」の「サービスにリンクされたロールの作成」を参照してください。

RUM のサービスにリンクされたロールの編集

CloudWatch RUM では、AWSServiceRoleForCloudWatchRUM ロールを編集することはできません。これらのロールは多くのエンティティにより参照されるため、ロールを作成した後その名前を変更することはできません。ただし、IAM を使用したロールの説明の編集はできます。

サービスにリンクされたロールの説明の編集 (IAM コンソール)

サービスにリンクされたロールの説明は、IAM コンソールを使用して編集できます。

サービスにリンクされたロールの説明を編集するには (コンソール)

1. IAM コンソールのナビゲーションペインで [ロール] を選択します。
2. 変更するロールの名前を選択します。
3. [Role description] の右端にある [Edit] を選択します。
4. ボックスに新しい説明を入力し、[Save] を選択します。

サービスにリンクされたロールの説明の編集 (AWS CLI)

AWS Command Line Interface から IAM コマンドを使用して、サービスにリンクされたロールの説明を編集できます。

サービスにリンクされたロールの説明を変更するには (AWS CLI)

1. (オプション) ロールの現在の説明を表示するには、次のコマンドを使用します。

   ```
   $ aws iam get-role --role-name role-name
   ```

   AWS CLI コマンドでは、ARN ではなくロール名を使用してロールを参照します。例えば、ロールの ARN が `arn:aws:iam::123456789012:role/myrole` である場合、そのロールを **`myrole`** と参照します。
2. サービスにリンクされたロールの説明を更新するには、次のコマンドを使用します。

   ```
   $ aws iam update-role-description --role-name role-name --description description
   ```

サービスにリンクされたロールの説明の編集 (IAM API)

サービスにリンクされたロールの説明は、IAM API を使用して編集できます。

サービスにリンクされたロールの説明を変更するには (API)

1. (オプション) ロールの現在の説明を表示するには、次のコマンドを使用します。

   GetRole
2. ロールの説明を更新するには、次のコマンドを使用します。

   UpdateRoleDescription

RUM のサービスにリンクされたロールの削除

X-Ray が有効になっているアプリケーションモニターが不要になった場合は、AWSServiceRoleForCloudWatchRUM ロールを削除することをお勧めします。

そうすることで、使用していないエンティティがアクティブにモニターリングされたり、メンテナンスされたりすることがなくなります。ただし、削除する前に、サービスにリンクされた役割をクリーンアップする必要があります。

サービスにリンクされたロールのクリーンアップ

IAM を使用してサービスにリンクされたロールを削除するには、まずそのロールにアクティブなセッションがないことを確認し、そのロールで使用されているリソースをすべて削除する必要があります。

サービスにリンクされたロールにアクティブなセッションがあるかどうかを、IAM コンソールで確認するには

1. IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
2. ナビゲーションペインで Roles (ロール) を選択します。AWSServiceRoleForCloudWatchRUM ロールの名前 (チェックボックスではない) を選択します。
3. 選択したロールの [概要] ページで [アクセスアドバイザー] を選択し、サービスにリンクされたロールの最新のアクティビティを確認します。

> **Note**
>
> RUM が AWSServiceRoleForCloudWatchRUM ロールを使用しているかどうか不明な場合は、ロールを削除してみてください。サービスでロールが使用されている場合、削除は失敗し、ロールが使用されている リージョンが表示されます。ロールが使用されてい

る場合は、ロールを削除する前にセッションが終了するのを待つ必要があります。サービスにリンクされたロールのセッションを取り消すことはできません。

サービスにリンクされたロールの削除 (IAM コンソール)

IAM コンソールを使用して、サービスにリンクされたロールを削除できます。

サービスにリンクされたロールを削除するには (コンソール)

1. IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
2. ナビゲーションペインで [Roles (ロール)] を選択します。ロール名または行そのものではなく、削除するロールの名前の横にあるチェックボックスをオンにします。
3. [ロールのアクション] で、[ロールの削除] を選択します。
4. 確認ダイアログボックスで、サービスの最終アクセス時間データを確認します。これは、選択したそれぞれのロールの AWS サービスへの最終アクセス時間を示します。これは、そのロールが現在アクティブであるかどうかを確認するのに役立ちます。続行するには、[はい、削除します] を選択します。
5. IAM コンソール通知を見て、サービスにリンクされたロールの削除の進行状況を監視します。IAM サービスにリンクされたロールの削除は非同期であるため、削除するロールを送信すると、削除タスクは成功または失敗する可能性があります。タスクが失敗した場合は、通知から [詳細を表示] または [リソースを表示] を選択して、削除が失敗した理由を知ることができます。そのロールで使用中のリソースがサービスにあるために削除に失敗した場合、この失敗の理由にはリソースのリストも含まれます。

サービスにリンクされたロールの削除 (AWS CLI)

AWS Command Line Interface から IAM コマンドを使用して、サービスにリンクされたロールを削除できます。

サービスにリンクされたロールを削除するには (AWS CLI)

1. サービスにリンクされているロールは、使用されている、または関連するリソースがある場合は削除できないため、削除リクエストを送信する必要があります。これらの条件が満たされない場合、そのリクエストは拒否される可能性があります。レスポンスから `deletion-task-id` を取得して、削除タスクのステータスを確認する必要があります。サービスにリンクされたロールの削除リクエストを送信するには、次のコマンドを入力します。

```
$ aws iam delete-service-linked-role --role-name service-linked-role-name
```

2. 削除タスクのステータスを確認するには、次のコマンドを入力します。

```
$ aws iam get-service-linked-role-deletion-status --deletion-task-id deletion-task-id
```

削除タスクのステータスは、NOT_STARTED、IN_PROGRESS、SUCCEEDED、または FAILED となります。削除が失敗した場合は、失敗した理由がコールによって返され、トラブルシューティングが可能になります。

サービスにリンクされたロールの削除 (IAM API)

IAM API を使用して、サービスにリンクされたロールを削除できます。

サービスにリンクされたロールを削除するには (API)

1. サービスにリンクされたロールの削除リクエストを送信するには、DeleteServiceLinkedRole を呼び出します。リクエストで、削除するロール名を指定します。

   サービスにリンクされているロールは、使用されている、または関連するリソースがある場合は削除できないため、削除リクエストを送信する必要があります。これらの条件が満たされない場合、そのリクエストは拒否される可能性があります。レスポンスから DeletionTaskId を取得して、削除タスクのステータスを確認する必要があります。

2. 削除タスクのステータスを確認するには、GetServiceLinkedRoleDeletionStatus を呼び出します。リクエストで DeletionTaskId を指定します。

   削除タスクのステータスは、NOT_STARTED、IN_PROGRESS、SUCCEEDED、または FAILED となります。削除が失敗した場合は、失敗した理由がコールによって返され、トラブルシューティングが可能になります。

CloudWatch Application Insights のサービスにリンクされたロールの使用

CloudWatch Application Insights は AWS Identity and Access Management (IAM) サービスにリンクされたロールを使用します。サービスにリンクされたロールは、CloudWatch Application Insights に直接リンクされた一意のタイプの IAM ロールです。サービスにリンクされたロールは、CloudWatch

Application Insights によって事前定義されており、お客様の代わりにサービスから AWS の他のサービスを呼び出す必要のある許可がすべて含まれています。

サービスにリンクされたロールを使用すると、必要なアクセス許可を手動で追加する必要がなくなるため、CloudWatch Application Insights の設定が簡単になります。CloudWatch Application Insights は、サービスにリンクされたロールのアクセス許可を定義します。特に定義されていない限り、CloudWatch Application Insights のみがロールを引き受けることができます。定義される許可は、信頼ポリシーと許可ポリシーに含まれており、その許可ポリシーを他の IAM エンティティにアタッチすることはできません。

サービスにリンクされたロールをサポートする他のサービスについては、「IAM と連携するAWS サービス」を参照して、サービスにリンクされたロール列がはいになっているサービスを見つけてください。あり、のリンクをクリックすると、該当するサービスにリンクされたロールに関するドキュメントを表示できます。

CloudWatch Application Insights のサービスにリンクされたロールのアクセス許可

CloudWatch Application Insights は、AWSServiceRoleForApplicationInsights という名前のサービスにリンクされたロールを使用します。Application Insights は、このロールを使用して、顧客のリソースグループの分析、メトリクスにアラームを作成する CloudFormation スタックの作成、EC2 インスタンスでの CloudWatch Agent の設定などの操作を実行します。このサービスリンクロールには、`CloudwatchApplicationInsightsServiceLinkedRolePolicy` という名前の IAM ポリシーがアタッチされています。このポリシーの更新については、「AWS マネージドポリシーに対する Application Insights の更新」を参照してください

ロールのアクセス許可ポリシーは、リソースに対して以下のアクションを完了することを CloudWatch Application Insights に許可します。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "cloudwatch:DescribeAlarmHistory",
        "cloudwatch:DescribeAlarms",
        "cloudwatch:GetMetricData",
        "cloudwatch:ListMetrics",
        "cloudwatch:PutMetricAlarm",
        "cloudwatch:DeleteAlarms",
        "cloudwatch:PutAnomalyDetector",
```

```
        "cloudwatch:DeleteAnomalyDetector",
        "cloudwatch:DescribeAnomalyDetectors"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "logs:FilterLogEvents",
        "logs:GetLogEvents",
        "logs:DescribeLogStreams",
        "logs:DescribeLogGroups"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "events:DescribeRule"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "cloudFormation:CreateStack",
        "cloudFormation:UpdateStack",
        "cloudFormation:DeleteStack",
        "cloudFormation:DescribeStackResources"
      ],
      "Resource": [
        "arn:aws:cloudformation:*:*:stack/ApplicationInsights-*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "cloudFormation:DescribeStacks",
```

```
        "cloudFormation:ListStackResources",
        "cloudFormation:ListStacks"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "tag:GetResources"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "resource-groups:ListGroupResources",
        "resource-groups:GetGroupQuery",
        "resource-groups:GetGroup"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "resource-groups:CreateGroup",
        "resource-groups:DeleteGroup"
      ],
      "Resource": [
        "arn:aws:resource-groups:*:*:group/ApplicationInsights-*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "elasticloadbalancing:DescribeLoadBalancers",
        "elasticloadbalancing:DescribeTargetGroups",
        "elasticloadbalancing:DescribeTargetHealth"
      ],
```

```
      "Resource": [
        "*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "autoscaling:DescribeAutoScalingGroups"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "ssm:PutParameter",
        "ssm:DeleteParameter",
        "ssm:AddTagsToResource",
        "ssm:RemoveTagsFromResource",
        "ssm:GetParameters"
      ],
      "Resource": "arn:aws:ssm:*:*:parameter/AmazonCloudWatch-ApplicationInsights-*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "ssm:CreateAssociation",
        "ssm:UpdateAssociation",
        "ssm:DeleteAssociation",
        "ssm:DescribeAssociation"
      ],
      "Resource": [
        "arn:aws:ec2:*:*:instance/*",
        "arn:aws:ssm:*:*:association/*",
        "arn:aws:ssm:*:*:managed-instance/*",
        "arn:aws:ssm:*:*:document/AWSEC2-
ApplicationInsightsCloudwatchAgentInstallAndConfigure",
        "arn:aws:ssm:*:*:document/AWS-ConfigureAWSPackage",
        "arn:aws:ssm:*:*:document/AmazonCloudWatch-ManageAgent"
      ]
    },
    {
      "Effect": "Allow",
```

```
      "Action": [
        "ssm:GetOpsItem",
        "ssm:CreateOpsItem",
        "ssm:DescribeOpsItems",
        "ssm:UpdateOpsItem",
        "ssm:DescribeInstanceInformation"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "ssm:AddTagsToResource"
      ],
      "Resource": "arn:aws:ssm:*:*:opsitem/*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "ssm:ListCommandInvocations",
        "ssm:GetCommandInvocation"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": "ssm:SendCommand",
      "Resource": [
        "arn:aws:ec2:*:*:instance/*",
        "arn:aws:ssm:*:*:document/AWSEC2-CheckPerformanceCounterSets",
        "arn:aws:ssm:*:*:document/AWS-ConfigureAWSPackage",
        "arn:aws:ssm:*:*:document/AWSEC2-DetectWorkload",
        "arn:aws:ssm:*:*:document/AmazonCloudWatch-ManageAgent"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "ec2:DescribeInstances",
        "ec2:DescribeVolumes",
```

```
        "ec2:DescribeVolumeStatus",
        "ec2:DescribeVpcs",
        "ec2:DescribeVpcAttribute",
        "ec2:DescribeNatGateways"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "rds:DescribeDBInstances",
        "rds:DescribeDBClusters"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "lambda:ListFunctions",
        "lambda:GetFunctionConfiguration",
        "lambda:ListEventSourceMappings"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "events:PutRule",
        "events:PutTargets",
        "events:RemoveTargets",
        "events:DeleteRule"
      ],
      "Resource": [
        "arn:aws:events:*:*:rule/AmazonCloudWatch-ApplicationInsights-*"
      ]
    },
    {
      "Effect": "Allow",
```

```
      "Action": [
        "xray:GetServiceGraph",
        "xray:GetTraceSummaries",
        "xray:GetTimeSeriesServiceStatistics",
        "xray:GetTraceGraph"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "dynamodb:ListTables",
        "dynamodb:DescribeTable",
        "dynamodb:DescribeContributorInsights",
        "dynamodb:DescribeTimeToLive"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "application-autoscaling:DescribeScalableTargets"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "s3:ListAllMyBuckets",
        "s3:GetMetricsConfiguration",
        "s3:GetReplicationConfiguration"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Effect": "Allow",
```

```
      "Action": [
        "states:ListStateMachines",
        "states:DescribeExecution",
        "states:DescribeStateMachine",
        "states:GetExecutionHistory"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "apigateway:GET"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "ecs:DescribeClusters",
        "ecs:DescribeContainerInstances",
        "ecs:DescribeServices",
        "ecs:DescribeTaskDefinition",
        "ecs:DescribeTasks",
        "ecs:DescribeTaskSets",
        "ecs:ListClusters",
        "ecs:ListContainerInstances",
        "ecs:ListServices",
        "ecs:ListTasks"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "ecs:UpdateClusterSettings"
      ],
      "Resource": [
        "arn:aws:ecs:*:*:cluster/*"
```

```
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "eks:DescribeCluster",
        "eks:DescribeFargateProfile",
        "eks:DescribeNodegroup",
        "eks:ListClusters",
        "eks:ListFargateProfiles",
        "eks:ListNodegroups",
        "fsx:DescribeFileSystems",
        "fsx:DescribeVolumes"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "sns:GetSubscriptionAttributes",
        "sns:GetTopicAttributes",
        "sns:GetSMSAttributes",
        "sns:ListSubscriptionsByTopic",
        "sns:ListTopics"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "sqs:ListQueues"
      ],
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "logs:DeleteSubscriptionFilter"
      ],
      "Resource": [
```

```
        "arn:aws:logs:*:*:log-group:*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "logs:PutSubscriptionFilter"
      ],
      "Resource": [
        "arn:aws:logs:*:*:log-group:*",
        "arn:aws:logs:*:*:destination:AmazonCloudWatch-ApplicationInsights-
LogIngestionDestination*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "elasticfilesystem:DescribeFileSystems"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "route53:GetHostedZone",
        "route53:GetHealthCheck",
        "route53:ListHostedZones",
        "route53:ListHealthChecks",
        "route53:ListQueryLoggingConfigs"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "route53resolver:ListFirewallRuleGroupAssociations",
        "route53resolver:GetFirewallRuleGroup",
        "route53resolver:ListFirewallRuleGroups",
        "route53resolver:ListResolverEndpoints",
        "route53resolver:GetResolverQueryLogConfig",
```

```
        "route53resolver:ListResolverQueryLogConfigs",
        "route53resolver:ListResolverQueryLogConfigAssociations",
        "route53resolver:GetResolverEndpoint",
        "route53resolver:GetFirewallRuleGroupAssociation"
      ],
      "Resource": [
        "*"
      ]
    }
  ]
}
```

サービスにリンクされたロールの作成、編集、削除をIAM エンティティ (ユーザー、グループ、ロールなど) に許可するには、許可を設定する必要があります。詳細については、IAM ユーザーガイドの「サービスにリンクされたロールの許可」を参照してください。

CloudWatch Application Insights 用のサービスにリンクされたロールの作成

サービスにリンクされたロールを手動で作成する必要はありません。AWS Management Console で新しい Application Insights アプリケーションを作成すると、CloudWatch Application Insights によってサービスにリンクされたロールが作成されます。

このサービスにリンクされたロールを削除した後で再度作成する必要が生じた場合は、同じプロセスを使用してアカウントにロールを再作成できます。新しい Application Insights アプリケーションを作成すると、CloudWatch Application Insights によってサービスにリンクされたロールが再び作成されます。

CloudWatch Application Insights のサービスにリンクされたロールの編集

CloudWatch Application Insights では、AWSServiceRoleForApplicationInsights のサービスにリンクされたロールを編集することはできません。サービスにリンクされたロールを作成すると、多くのエンティティによってロールが参照される可能性があるため、ロール名を変更することはできません。ただし、IAM を使用したロールの説明の編集はできます。詳細については、IAM ユーザーガイドの「サービスにリンクされたロールの編集」を参照してください。

CloudWatch Application Insights のサービスにリンクされたロールの削除

サービスにリンクされたロールが必要な機能またはサービスが不要になった場合には、そのロールを削除することをお勧めします。そうすることで、使用していないエンティティがアクティブにモニタリングまたはメンテナンスされることがなくなります。ただし、ロールを手動で削除する前に、Application Insights のすべてのアプリケーションを削除する必要があります。

> **Note**
>
> リソースを削除する際に、CloudWatch Application Insights のサービスでロールが使用されている場合、削除は失敗することがあります。失敗した場合は、数分待ってから再度オペレーションを実行してください。

AWSServiceRoleForApplicationInsights で使用される CloudWatch Application Insights リソースを削除するには

- CloudWatch Application Insights アプリケーションをすべて削除します。詳細については、CloudWatch Application Insights ユーザーガイドの「アプリケーションの削除」を参照してください。

IAM を使用してサービスリンクロールを手動で削除するには

IAM コンソール、AWS CLI、または AWS API を使用して、AWSServiceRoleForApplicationInsights のサービスにリンクされたロールを削除します。詳細については、IAM ユーザーガイドの「サービスにリンクされたロールの削除」を参照してください。

CloudWatch Application Insights のサービスにリンクされたロールがサポートされるリージョン

CloudWatch Application Insights は、そのサービスを利用できるすべての AWS リージョンで、サービスにリンクされたロールの使用をサポートします。詳細については、「CloudWatch Application Insights リージョンとエンドポイント」を参照してください。

Amazon CloudWatch Application Insights の AWS マネージドポリシー

AWS マネージドポリシーは、AWS が作成および管理するスタンドアロンポリシーです。AWS マネージドポリシーは、多くの一般的なユースケースでアクセス許可を提供できるように設計されているため、ユーザー、グループ、ロールへのアクセス許可の割り当てを開始できます。

AWS マネージドポリシーは、ご利用の特定のユースケースに対して最小特権のアクセス許可を付与しない場合があることにご注意ください。AWS のすべてのお客様が使用できるようになるのを避け

るためです。ユースケース別にカスタマー管理ポリシーを定義することで、アクセス許可を絞り込むことをお勧めします。

AWS 管理ポリシーで定義したアクセス権限は変更できません。AWS が AWS マネージドポリシーに定義されているアクセス許可を更新すると、更新はポリシーがアタッチされているすべてのプリンシパルアイデンティティ (ユーザー、グループ、ロール) に影響します。新しい AWS のサービスを起動するか、既存のサービスで新しい API オペレーションが使用可能になると、AWS が AWS マネージドポリシーを更新する可能性が最も高くなります。

詳細については、「IAM ユーザーガイド」の「AWS 管理ポリシー」を参照してください。

AWS マネージドポリシー: CloudWatchApplicationInsightsFullAccess

`CloudWatchApplicationInsightsFullAccess` ポリシーは IAM ID にアタッチできます。

このポリシーにより、Application Insights 機能へのフルアクセスを許可する管理者権限が付与されます。

許可の詳細

このポリシーには以下のアクセス許可が含まれています。

- `applicationinsights` – Application Insights 機能へのフルアクセスを許可します。
- `iam` – Application Insights で、サービスにリンクされたロール AWSServiceRoleForApplicationInsights の作成を許可します。これは、Application Insights が顧客のリソースグループの分析、メトリクスにアラームを作成する CloudFormation スタックの作成、EC2 インスタンスでの CloudWatch Agent の設定などの操作を実行できるようにするために必要です。詳細については、「CloudWatch Application Insights のサービスにリンクされたロールの使用」を参照してください。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "applicationinsights:*",
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "ec2:DescribeInstances",
        "ec2:DescribeVolumes",
        "rds:DescribeDBInstances",
        "rds:DescribeDBClusters",
        "sqs:ListQueues",
        "elasticloadbalancing:DescribeLoadBalancers",
        "elasticloadbalancing:DescribeTargetGroups",
        "elasticloadbalancing:DescribeTargetHealth",
        "autoscaling:DescribeAutoScalingGroups",
        "lambda:ListFunctions",
        "dynamodb:ListTables",
        "s3:ListAllMyBuckets",
        "sns:ListTopics",
        "states:ListStateMachines",
        "apigateway:GET",
        "ecs:ListClusters",
        "ecs:DescribeTaskDefinition",
        "ecs:ListServices",
        "ecs:ListTasks",
        "eks:ListClusters",
        "eks:ListNodegroups",
        "fsx:DescribeFileSystems",
        "logs:DescribeLogGroups",
        "elasticfilesystem:DescribeFileSystems"
      ],
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "iam:CreateServiceLinkedRole"
      ],
```

```
      "Resource": [
        "arn:aws:iam::*:role/aws-service-role/application-insights.amazonaws.com/
AWSServiceRoleForApplicationInsights"
      ],
      "Condition": {
        "StringEquals": {
          "iam:AWSServiceName": "application-insights.amazonaws.com"
        }
      }
    }
  ]
}
```

AWS マネージドポリシー: CloudWatchApplicationInsightsReadOnlyAccess

`CloudWatchApplicationInsightsReadOnlyAccess` ポリシーは IAM ID にアタッチできます。

このポリシーにより、Application Insights 機能への読み取り専用アクセスを許可する管理者権限が付与されます。

許可の詳細

このポリシーには以下のアクセス許可が含まれています。

- `applicationinsights` – Application Insights 機能への読み取り専用アクセスを許可します。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "applicationinsights:Describe*",
                "applicationinsights:List*"
            ],
            "Resource": "*"
        }
    ]
```

```
}
```

AWS マネージドポリシー: CloudwatchApplicationInsightsServiceLinkedRolePolicy

CloudwatchApplicationInsightsServiceLinkedRolePolicy を IAM エンティティにアタッチすることはできません。このポリシーは、Application Insights が顧客リソースを監視できるようにするサービスにリンクされたロールにアタッチされます。詳細については、「CloudWatch Application Insights のサービスにリンクされたロールの使用」を参照してください。

AWS マネージドポリシーに対する Application Insights の更新

Application Insights の AWS マネージドポリシーの更新に関する詳細を、このサービスがこれらの変更の追跡を開始した以降の分について表示します。このページの変更に関する自動通知を入手するには、Application Insights ドキュメントの履歴ページから、RSS フィードにサブスクライブしてください。

変更	説明	日付
CloudwatchApplicationInsightsServiceLinkedRolePolicy – 既存のポリシーの更新	Application Insights に、CloudFormation スタックを一覧表示するための新しい許可が追加されました。 これらの許可は、Amazon CloudWatch Application Insights が CloudFormation スタックにネストされている AWS リソースを分析し、モニタリングするために必要です。	2023 年 4 月 24 日
CloudwatchApplicationInsightsServiceLinkedRolePolicy – 既存のポリシーの更新	Application Insights が、Amazon VPC と Route 53 リソースのリストを取得する	2023 年 1 月 23 日

変更	説明	日付
	ための新しい許可を追加しました。 これらの許可は、Amazon CloudWatch Applicati on Insights が Amazon CloudWatch でベストプラクティスのネットワークモニタリングを自動的にセットアップするために必要です。	
CloudwatchApplicationInsightsServiceLinkedRolePolicy – 既存のポリシーの更新	Application Insights が、SSM コマンドの呼び出し結果を取得するための新しい許可を追加しました。 これらの許可は、Amazon CloudWatch Application Insights が Amazon EC2 インスタンスで実行されているワークロードを自動的に検出し、モニタリングするために必要です。	2022 年 12 月 19 日

変更	説明	日付
CloudwatchApplicationInsightsServiceLinkedRolePolicy – 既存のポリシーの更新	Application Insights が、Amazon VPC および Route 53 リソースを記述するための新しい許可を追加しました。 この許可は、Amazon CloudWatch Application Insights がお客様の Amazon VPC および Route 53 リソース設定を読み取り、Amazon CloudWatch でベストプラクティスのネットワークモニタリングを自動的にセットアップできるようにするために必要です。	2022 年 12 月 19 日
CloudwatchApplicationInsightsServiceLinkedRolePolicy – 既存のポリシーの更新	Application Insights が、EFS リソースを記述するための新しい許可を追加しました。 この許可は、Amazon CloudWatch Application Insights がお客様の Amazon EFS リソースの設定を読み取り、CloudWatch で EFS モニタリングのベストプラクティスを自動的にセットアップできるようにするために必要です。	2022 年 10 月 3 日

変更	説明	日付
CloudwatchApplicationInsightsServiceLinkedRolePolicy – 既存のポリシーの更新	Application Insights が、EFS ファイルシステムを記述するための新しい許可を追加しました。 これらのアクセス許可は、Amazon CloudWatch Application Insights がアカウント内のサポートされているすべてのリソースをクエリしてアカウントベースのアプリケーションを作成するために必要です。	2022 年 10 月 3 日
CloudwatchApplicationInsightsServiceLinkedRolePolicy – 既存のポリシーの更新	Application Insights が、FSx リソースに関する情報を取得するための新しい許可を追加しました。 これらの許可は、Amazon CloudWatch Application Insights が基盤となる FSx ボリュームに関する十分な情報を取得することで、ワークロードをモニタリングするために必要です。	2022 年 9 月 12 日

変更	説明	日付
AWS マネージドポリシー: CloudWatchApplicationInsightsFullAccess – 既存ポリシーへの更新。	Application Insights がロググループを記述するための新しいアクセス許可を追加しました。 このアクセス許可は、Amazon CloudWatch Application Insights で、新しいアプリケーションを作成するときにロググループを監視するための正しいアクセス許可がアカウント内にあることを確認するために必要です。	2022 年 1 月 24 日
CloudwatchApplicationInsightsServiceLinkedRolePolicy – 既存のポリシーの更新	Application Insights が CloudWatch Logs サブスクリプションフィルターを作成および削除するための新しいアクセス許可を追加しました。 これらのアクセス許可は、Amazon CloudWatch Application Insights がサブスクリプションフィルターを作成して、設定済みのアプリケーション内のリソースのログモニタリングを容易にするために必要です。	2022 年 1 月 24 日

変更	説明	日付
CloudwatchApplicationInsightsServiceLinkedRolePolicy – 既存のポリシーの更新	Application Insights では、Elastic Load Balancer のターゲットグループとターゲットヘルスを記述する新しいアクセス許可が追加されました。 これらのアクセス許可は、Amazon CloudWatch Application Insights がアカウント内のサポートされているすべてのリソースをクエリしてアカウントベースのアプリケーションを作成するために必要です。	2021 年 11 月 4 日
CloudwatchApplicationInsightsServiceLinkedRolePolicy – 既存のポリシーの更新	Application Insights では、Amazon EC2 インスタンスで `AmazonCloudWatch-ManageAgent` SSM ドキュメントを実行する新しいアクセス許可が追加されました。 このアクセス許可は、Amazon CloudWatch Application Insights が Application Insights で作成した CloudWatch エージェント設定ファイルをクリーンアップするために必要です。	2021 年 9 月 30 日

変更	説明	日付
CloudwatchApplicationInsightsServiceLinkedRolePolicy – 既存のポリシーの更新	Application Insights では、アカウントベースのアプリケーションモニタリングをサポートする新しいアクセス許可が追加され、アカウント内のサポートされているすべてのリソースをオンボードおよびモニタリングできるようになりました。 これらのアクセス許可は、Amazon CloudWatch Application Insights がクエリ、リソースのタグ付け、これらのリソースのグループの作成を行うのに必要です。 Application Insights では、SNS トピックのモニタリングをサポートする新しいアクセス許可が追加されました。 これらのアクセス許可は、Amazon CloudWatch Application Insights が SNS リソースからメタデータを収集し、SNS トピックのモニタリングを設定するために必要です。	2021 年 9 月 15 日

変更	説明	日付
AWS マネージドポリシー: CloudWatchApplicationInsightsFullAccess – 既存ポリシーへの更新。	Application Insights では、サポートされているリソースを記述して一覧表示するための新しいアクセス許可が追加されました。 これらのアクセス許可は、Amazon CloudWatch Application Insights がアカウント内のサポートされているすべてのリソースをクエリしてアカウントベースのアプリケーションを作成するために必要です。	2021 年 9 月 15 日
CloudwatchApplicationInsightsServiceLinkedRolePolicy – 既存のポリシーの更新	Application Insights では、FSx リソースを記述するための新しいアクセス許可が追加されました。 このアクセス許可は、Amazon CloudWatch Application Insights がお客様の FSx リソースの設定を読み取り、CloudWatch によってベストプラクティスの FSx モニタリングを自動的にセットアップできるようにするために必要なものです。	2021 年 8 月 31 日

変更	説明	日付
CloudwatchApplicationInsightsServiceLinkedRolePolicy – 既存のポリシーの更新	Application Insights では、ECS および EKS サービスリソースについて説明および一覧表示する新しいアクセス許可が追加されました。 このアクセス許可は、Amazon CloudWatch Application Insights がカスタマーコンテナリソースの設定を読み取り、CloudWatch でベストプラクティスのコンテナモニタリングを自動的にセットアップできるようにするために必要です。	2021 年 5 月 18 日
CloudwatchApplicationInsightsServiceLinkedRolePolicy – 既存のポリシーの更新	Application Insights は、OpsCenter が `opsitem` リソースタイプを持つリソース上の `ssm:AddTagsToResource` アクションを使用して OpsItems をタグ付けできるようにする新しいアクセス許可を追加しました。 このアクセス許可は OpsCenter で必要です。Amazon CloudWatch Application Insights は、お客様が AWS SSM OpsCenter を使用して問題を解決できるように OpsItems を作成します。	2021 年 4 月 13 日

変更	説明	日付
Application Insights が変更の追跡を開始しました	Application Insights は、AWS マネージドポリシーの変更の追跡を開始しました。	2021 年 4 月 13 日

Amazon CloudWatch の許可リファレンス

次の表は、各 CloudWatch API オペレーションや、そのアクションを実行するためにアクセス許可を付与できる関連アクションを示しています。ポリシーの `Action` フィールドでアクションを指定し、ポリシーの `Resource` フィールドでリソース値としてワイルドカード文字 (*) を指定します。

CloudWatch ポリシーで AWS 全体の条件キーを使用して、条件を表現できます。AWS 全体を対象とするすべてのキーのリストについては、IAM ユーザーガイドの AWS グローバルキーと IAM 条件コンテキストキーを参照してください。

> **Note**
>
> アクションを指定するには、API オペレーション名の前に `cloudwatch:` プレフィックスを使用します。例えば、`cloudwatch:GetMetricData`、`cloudwatch:ListMetrics`、または `cloudwatch:*` (すべての CloudWatch アクションの場合)。

トピック

CloudWatch API オペレーションおよびアクションに必要な許可

CloudWatch API オペレーション	必要なアクセス許可 (API アクション)

CloudWatch API オペレーション	必要なアクセス許可 (API アクション)
DeleteAlarms	`cloudwatch:DeleteAlarms` アラームを削除するのに必要です。
DeleteDashboards	`cloudwatch:DeleteDashboards` ダッシュボードを削除するために必要です。
DeleteMetricStream	`cloudwatch:DeleteMetricStream` メトリクスストリームを削除するために必要です。
DescribeAlarmHistory	`cloudwatch:DescribeAlarmHistory` アラーム履歴を表示するのに必要です。複合アラームに関する情報を取得するには、`cloudwatch:DescribeAlarmHistory` のアクセス許可に * の範囲が必要です。`cloudwatch:DescribeAlarmHistory` のアクセス許可の範囲が狭い場合、複合アラームに関する情報を返すことはできません。
DescribeAlarms	`cloudwatch:DescribeAlarms` アラームに関する情報を取得するために必要です。 複合アラームに関する情報を取得するには、`cloudwatch:DescribeAlarms` のアクセス許可に * の範囲が必要です。`cloudwatch:DescribeAlarms` のアクセス許可の範囲が狭い場合、複合アラームに関する情報を返すことはできません。

CloudWatch API オペレーション	必要なアクセス許可 (API アクション)
DescribeAlarmsForMetric	`cloudwatch:DescribeAlarmsForMetric` メトリクスのアラームを表示するのに必要です。
DisableAlarmActions	`cloudwatch:DisableAlarmActions` アラームアクションを無効にするのに必要です。
EnableAlarmActions	`cloudwatch:EnableAlarmActions` アラームアクションを有効にするのに必要です。
GetDashboard	`cloudwatch:GetDashboard` 既存のダッシュボードに関するデータを表示するのに必要です。
GetMetricData	`cloudwatch:GetMetricData` CloudWatch コンソールでメトリクスデータをグラフ化し、大規模なメトリクスデータのバッチを取得してそのデータで Metric Math を実行するために必要となるものです。
GetMetricStatistics	`cloudwatch:GetMetricStatistics` CloudWatch コンソールの他の部分やダッシュボードウィジェットにあるグラフを見るのに必要です。

CloudWatch API オペレーション	必要なアクセス許可 (API アクション)
GetMetricStream	`cloudwatch:GetMetricStream` メトリクスストリームに関する情報を表示するために必要です。
GetMetricWidgetImage	`cloudwatch:GetMetricWidgetImage` 1 つ以上の CloudWatch メトリクスのスナップショットグラフをビットマップイメージとして取得するために必要です。
ListDashboards	`cloudwatch:ListDashboards` アカウントで CloudWatch ダッシュボードのリストを表示するのに必要です。
ListMetrics	`cloudwatch:ListMetrics` CloudWatch コンソール内または CLI のメトリクス名を表示または検索するのに必要です。ダッシュボードウィジェットでメトリクスを選択するために必要です。
ListMetricStreams	`cloudwatch:ListMetricStreams` アカウント内のメトリクスストリームのリストを、表示または検索するために必要です。

CloudWatch API オペレーション	必要なアクセス許可 (API アクション)
PutCompositeAlarm	`cloudwatch:PutCompositeAlarm` 複合アラームを作成するために必要です。 複合アラームを作成するには、`cloudwatch:PutCompositeAlarm` のアクセス許可に * の範囲が必要です。`cloudwatch:PutCompositeAlarm` のアクセス許可の範囲が狭い場合、複合アラームに関する情報を返すことはできません。
PutDashboard	`cloudwatch:PutDashboard` ダッシュボードを作成または既存のダッシュボードを更新するのに必要です。
PutMetricAlarm	`cloudwatch:PutMetricAlarm` アラームを作成または更新するのに必要です。
PutMetricData	`cloudwatch:PutMetricData` メトリクスを作成するために必要です。
PutMetricStream	`cloudwatch:PutMetricStream` メトリクスストリームを作成するために必要です。
SetAlarmState	`cloudwatch:SetAlarmState` 手動でアラームの状態を設定するのに必要です。

CloudWatch API オペレーション	必要なアクセス許可 (API アクション)
StartMetricStreams	`cloudwatch:StartMetricStreams` メトリクスストリームでメトリクスのフローを開始するために必要です。
StopMetricStreams	`cloudwatch:StopMetricStreams` メトリクスストリーム内のメトリクスのフローを、一時的に停止するために必要です。
TagResource	`cloudwatch:TagResource` アラームや Contributor Insights ルールなど、CloudWatch リソースのタグを追加または更新するために必要です。
UntagResource	`cloudwatch:UntagResource` CloudWatch リソースからタグを削除するために必要です。

CloudWatch Contributor Insights API オペレーションとアクションに必要な許可

Important

ユーザーに `cloudwatch:PutInsightRule` アクセス許可を付与すると、デフォルトでは、そのユーザーは CloudWatch Logs 内の任意のロググループを評価するルールを作成できます。特定のロググループを含める、および除外するユーザーのアクセス許可を制限する IAM ポリシー条件を追加できます。詳細については、「Contributor Insights のユーザーのロググループへのアクセスを制限するための条件キーの使用」を参照してください。

CloudWatch Contributor Insights API オペレーション	必要なアクセス許可 (API アクション)
DeleteInsightRules	`cloudwatch:DeleteInsightRules` Contributor Insights ルールを削除するために必要です。
DescribeInsightRules	`cloudwatch:DescribeInsightRules` アカウントの Contributor Insights ルールを表示するために必要です。
EnableInsightRules	`cloudwatch:EnableInsightRules` Contributor Insights ルールを有効にするために必要です。
GetInsightRuleReport	`cloudwatch:GetInsightRuleReport` Contributor Insights ルールによって収集された時系列データおよびその他の統計情報を取得するために必要です。
PutInsightRule	`cloudwatch:PutInsightRule` Contributor Insights ルールを作成するために必要です。このテーブルの冒頭にある「重要」の注記を参照してください。

CloudWatch Events API オペレーションおよびアクションに必要な許可

CloudWatch Events API オペレーション	必要なアクセス許可 (API アクション)

CloudWatch Events API オペレーション	必要なアクセス許可 (API アクション)
DeleteRule	`events:DeleteRule` ルールを削除するのに必要です。
DescribeRule	`events:DescribeRule` ルールについての詳細を一覧表示するのに必要です。
DisableRule	`events:DisableRule` ルールを無効にするのに必要です。
EnableRule	`events:EnableRule` ルールを有効にするのに必要です。
ListRuleNamesByTarget	`events:ListRuleNamesByTarget` ターゲットと関連付けられるルールを一覧表示するために必要です。
ListRules	`events:ListRules` アカウントの全グループを一覧表示するために必要です。
ListTargetsByRule	`events:ListTargetsByRule` ルールと関連付けられるすべてのターゲットを一覧表示するために必要です。

CloudWatch Events API オペレーション	必要なアクセス許可 (API アクション)
PutEvents	`events:PutEvents` ルールと一致するカスタムイベントを追加するために必要です。
PutRule	`events:PutRule` ルールを作成または更新するために必要です。
PutTargets	`events:PutTargets` ルールにターゲットを追加するために必要です。
RemoveTargets	`events:RemoveTargets` ターゲットをルールから削除するために必要です。
TestEventPattern	`events:TestEventPattern` 特定のイベントに対してイベントパターンをテストするために必要です。

CloudWatch Logs API オペレーションおよびアクションに必要な許可

CloudWatch Logs API のオペレーション	必要なアクセス許可 (API アクション)
CancelExportTask	`logs:CancelExportTask` 保留中または実行中のエクスポートタスクをキャンセルするのに必要です。

CloudWatch Logs API のオペレーション	必要なアクセス許可 (API アクション)
CreateExportTask	`logs:CreateExportTask` ロググループから Amazon S3バケットにデータをエクスポートするのに必要です。
CreateLogGroup	`logs:CreateLogGroup` 新しいロググループを作成するのに必要です。
CreateLogStream	`logs:CreateLogStream` ロググループに新しいログストリームを作成するのに必要です。
DeleteDestination	`logs:DeleteDestination` ログ宛先を削除したり、サブスクリプションのフィルタを無効化するのに必要です。
DeleteLogGroup	`logs:DeleteLogGroup` ロググループや関連したアーカイブログイベントを削除するのに必要です。
DeleteLogStream	`logs:DeleteLogStream` ログストリームや関連したアーカイブログイベントを削除するのに必要です。
DeleteMetricFilter	`logs:DeleteMetricFilter` ロググループに関連したメトリクスフィルタを削除するのに必要です。

CloudWatch Logs API のオペレーション	必要なアクセス許可 (API アクション)
DeleteQueryDefinition	`logs:DeleteQueryDefinition` CloudWatch Logs Insights で保存されたクエリ定義を削除するのに必要です。
DeleteResourcePolicy	`logs:DeleteResourcePolicy` CloudWatch Logs リソースポリシーを削除するために必要です。
DeleteRetentionPolicy	`logs:DeleteRetentionPolicy` ロググループの保持ポリシーを削除するのに必要です。
DeleteSubscriptionFilter	`logs:DeleteSubscriptionFilter` ロググループに関連したサブスクリプションのフィルタを削除するのに必要です。
DescribeDestinations	`logs:DescribeDestinations` アカウントに関連したすべての送信先を表示するのに必要です。
DescribeExportTasks	`logs:DescribeExportTasks` アカウントに関連したすべてのエクスポートタスクを表示するのに必要です。
DescribeLogGroups	`logs:DescribeLogGroups` アカウントに関連したすべてのロググループを表示するのに必要です。

CloudWatch Logs API のオペレーション	必要なアクセス許可 (API アクション)
DescribeLogStreams	`logs:DescribeLogStreams` ロググループに関連したすべてのログストリームを表示するのに必要です。
DescribeMetricFilters	`logs:DescribeMetricFilters` ロググループに関連したすべてのメトリクスを表示するのに必要です。
DescribeQueryDefinitions	`logs:DescribeQueryDefinitions` CloudWatch Logs Insights で保存されたクエリ定義のリストを表示するために必要です。
DescribeQueries	`logs:DescribeQueries` スケジュールされた、実行された、または最近実行された CloudWatch Logs Insights クエリのリストを表示するために必要です。
DescribeResourcePolicies	`logs:DescribeResourcePolicies` CloudWatch Logs リソースポリシーのリストを表示するために必要です。
DescribeSubscriptionFilters	`logs:DescribeSubscriptionFilters` ロググループに関連したすべてのサブスクリプションフィルタを表示するのに必要です。

CloudWatch Logs API のオペレーション	必要なアクセス許可 (API アクション)
FilterLogEvents	`logs:FilterLogEvents` ロググループのフィルタパターンでログイベントをソートするのに必要です。
GetLogEvents	`logs:GetLogEvents` ログストリームからログイベントを取得するのに必要です。
GetLogGroupFields	`logs:GetLogGroupFields` ロググループのログイベントに含まれるフィールドのリストを取得するために必要です。
GetLogRecord	`logs:GetLogRecord` 1 つのログイベントから詳細を取得するために必要です。
GetQueryResults	`logs:GetQueryResults` CloudWatch Logs Insights クエリの結果を取得するために必要です。
ListTagsLogGroup	`logs:ListTagsLogGroup` ロググループに関連したタグをリストするのに必要です。
PutDestination	`logs:PutDestination` 宛先ログストリーム (Kinesis ストリームなど) の作成や更新に必要です。

CloudWatch Logs API のオペレーション	必要なアクセス許可 (API アクション)
PutDestinationPolicy	`logs:PutDestinationPolicy` 既存の口グ宛先に関連するアクセスポリシーの作成や更新に必要です。
PutLogEvents	`logs:PutLogEvents` 一連のログイベントをログストリームに更新するのに必要です。
PutMetricFilter	`logs:PutMetricFilter` メトリクスフィルタの作成や更新、またはそれをロググループに関連付けるのに必要です。
PutQueryDefinition	`logs:PutQueryDefinition` CloudWatch Logs Insights にクエリを保存するために必要です。
PutResourcePolicy	`logs:PutResourcePolicy` CloudWatch Logs リソースポリシーを作成するために必要です。
PutRetentionPolicy	`logs:PutRetentionPolicy` ロググループでログイベント (保持) を維持する日数を設定するのに必要です。

CloudWatch Logs API のオペレーション	必要なアクセス許可 (API アクション)
PutSubscriptionFilter	`logs:PutSubscriptionFilter` サブスクリプションフィルタの作成や更新、またはそれをロググループに関連付けるのに必要です。
StartQuery	`logs:StartQuery` CloudWatch Logs Insights クエリを開始するために必要です。
StopQuery	`logs:StopQuery` 進行中の CloudWatch Logs Insights クエリを停止するために必要です。
TagLogGroup	`logs:TagLogGroup` ロググループのタグを追加または更新するのに必要です。
TestMetricFilter	`logs:TestMetricFilter` ログイベントメッセージのサンプリングに対してフィルタパターンをテストするのに必要です。

Amazon EC2 API オペレーションおよびアクションで必要な許可

Amazon EC2 API オペレーション	必要なアクセス許可 (API アクション)
DescribeInstanceStatus	`ec2:DescribeInstanceStatus`

Amazon EC2 API オペレーション	必要なアクセス許可 (API アクション)
	EC2 インスタンスの状態の詳細を表示するのに必要です。
DescribeInstances	`ec2:DescribeInstances` EC2 インスタンスの詳細を表示するのに必要です。
RebootInstances	`ec2:RebootInstances` EC2 インスタンスを再起動するために必要です。
StopInstances	`ec2:StopInstances` EC2 インスタンスを停止するのに必要です。
TerminateInstances	`ec2:TerminateInstances` EC2 インスタンスを削除するのに必要です。

Amazon EC2 Auto Scaling API オペレーションとアクションに必要な許可

Amazon EC2 Auto Scaling API オペレーション	必要なアクセス許可 (API アクション)
スケーリング	`autoscaling:Scaling` Auto Scaling グループをスケーリングするために必要です。
Trigger	`autoscaling:Trigger`

Amazon EC2 Auto Scaling API オペレーション	必要なアクセス許可 (API アクション)
	Auto Scaling アクションをトリガーするために必要です。

Amazon CloudWatch のコンプライアンス検証

サードパーティーの監査者は、さまざまな AWS コンプライアンスプログラムの一環として Amazon CloudWatch のセキュリティとコンプライアンスを評価します。このプログラムには、SOC、PCI、FedRAMP、HIPAA などがあります。

特定のコンプライアンスプログラムの対象となる AWS サービスのリストについては、「コンプライアンスプログラムによる AWS 対象範囲内のサービス」を参照してください。一般的な情報については、「AWS コンプライアンスプログラム」「」「」を参照してください。

AWS Artifact を使用して、サードパーティーの監査レポートをダウンロードできます。詳細については、「AWS Artifact 内のレポートのダウンロード」「」を参照してください。

Amazon CloudWatch を使用する際のお客様のコンプライアンス責任は、データの機密性、企業のコンプライアンス目的、適用法規によって決まります。AWS ではコンプライアンスに役立つ以下のリソースを用意しています。

- 「セキュリティとコンプライアンスのクイックスタートガイド」「」 – これらのデプロイガイドには、アーキテクチャ上の考慮事項の説明と、AWS でセキュリティとコンプライアンスに重点を置いたベースライン環境をデプロイするためのステップが記載されています。
- HIPAA セキュリティおよびコンプライアンスホワイトペーパーのアーキテクチャの設計 - このホワイトペーパーでは、企業が AWS を使用して HIPAA 準拠のアプリケーションを作成する方法について説明します。
- AWSコンプライアンスのリソース– このワークブックおよびガイドのコレクションは、お客様の業界と拠点に適用されるものである場合があります。
- AWS Configデベロッパーガイドのルールでのリソースの評価 – AWS Configは、リソース設定が、社内のプラクティス、業界のガイドラインそして規制にどの程度適合しているのかを評価します。
- AWS Security Hub– AWSのこのサービスは、 AWS内でのユーザーのセキュリティ状態に関する包括的な見解を提供し、業界のセキュリティ標準、およびベストプラクティスに対するコンプライアンスを確認するために役立ちます。

Amazon CloudWatch の耐障害性

AWS のグローバルインフラストラクチャは AWS リージョンとアベイラビリティーゾーンを中心として構築されます。リージョンには、低レイテンシー、高いスループット、そして高度の冗長ネットワークで接続されている複数の物理的に独立および隔離されたアベイラビリティーゾーンがあります。アベイラビリティーゾーンでは、ゾーン間で中断することなく自動的にフェイルオーバーするアプリケーションとデータベースを設計および運用することができます。アベイラビリティーゾーンは、従来の単一または複数のデータセンターインフラストラクチャよりも可用性、耐障害性、および拡張性が優れています。

AWS リージョンとアベイラビリティーゾーンの詳細については、AWS グローバルインフラストラクチャを参照してください。

Amazon CloudWatch のインフラストラクチャセキュリティ

マネージドサービスである Amazon CloudWatch は、AWS グローバルネットワークセキュリティによって保護されています。AWSセキュリティサービスと AWS がインフラストラクチャを保護する方法については、「AWS クラウドセキュリティ」を参照してください。インフラストラクチャセキュリティのベストプラクティスを使用して AWS 環境を設計するには、「セキュリティの柱 - AWS Well-Architected Framework」の「インフラストラクチャ保護」を参照してください。

AWS が発行した API 呼び出しを使用して、ネットワーク経由で CloudWatch にアクセスします。クライアントは以下をサポートする必要があります。

- Transport Layer Security (TLS) TLS 1.2 および TLS 1.3 をお勧めします。
- DHE (Ephemeral Diffie-Hellman) や ECDHE (Elliptic Curve Ephemeral Diffie-Hellman) などの Perfect Forward Secrecy (PFS) を使用した暗号スイートです。これらのモードは、Java 7 以降など、最近のほとんどのシステムでサポートされています。

また、リクエストは、アクセスキー ID と、IAM プリンシパルに関連付けられているシークレットアクセスキーを使用して署名する必要があります。または、AWS Security Token Service (AWS STS) を使用して、一時セキュリティ認証情報を生成し、リクエストに署名することもできます。

ネットワークの隔離

Virtual Private Cloud (VPC) は、Amazon Web Services クラウド内の論理的に隔離された領域にある仮想ネットワークです。サブネットは、ある範囲の IP アドレスが示す VPC 内の領域です。VPC の

サブネットに、さまざまな AWS リソースをデプロイできます。たとえば、サブネットに Amazon EC2 インスタンス、EMR クラスター、および DynamoDB テーブルをデプロイできます。詳細については、Amazon VPC ユーザーガイドを参照してください。

CloudWatch がパブリックインターネットを経由せずに VPC 内のリソースと通信できるようにするには、AWS PrivateLink を使用します。詳細については、「インターフェイス VPC エンドポイントでの CloudWatch および CloudWatch Synthetics の使用」を参照してください。

プライベートサブネットは、パブリックインターネットへのデフォルトルートを持たないサブネットです。プライベートサブネットに AWS リソースをデプロイしても、Amazon CloudWatch がリソースから組み込みメトリクスを収集するのを防ぐことはできません。

プライベートサブネットの AWS リソースからカスタムメトリクスを発行する必要がある場合は、プロキシサーバーを使用して発行できます。プロキシサーバーは、これらの HTTPS 要求を CloudWatch のパブリック API エンドポイントに転送します。

AWS Security Hub

AWS Security Hub を使用して、セキュリティのベストプラクティスに関連する CloudWatch の使用状況をモニタリングします。Security Hub は、セキュリティコントロールを使用してリソース設定とセキュリティ標準を評価し、お客様がさまざまなコンプライアンスフレームワークに準拠できるようサポートします。Security Hub を使用して CloudWatch リソースを評価する方法の詳細については、「AWS Security Hub ユーザーガイド」の「Amazon CloudWatch コントロール」を参照してください。

インターフェイス VPC エンドポイントでの CloudWatch および CloudWatch Synthetics の使用

Amazon Virtual Private Cloud (Amazon VPC) を使用して AWS リソースをホストする場合、VPC、CloudWatch、および CloudWatch Synthetics の間のプライベート接続を確立できます。この接続を使用すると、CloudWatch と CloudWatch Synthetics はパブリックインターネットを経由せずに VPC のリソースと通信できます。

Amazon VPC は、ユーザー定義の仮想ネットワークで AWS リソースを起動するために使用できる AWS のサービスです。VPC を使用すると、IP アドレス範囲、サブネット、ルートテーブル、ネットワークゲートウェイなどのネットワーク設定を制御できます。VPC を CloudWatch または CloudWatch Synthetics に接続するには、VPC を AWS のサービスに接続するためのインターフェ

イス VPC エンドポイントを定義します。このエンドポイントを使用すると、インターネットゲートウェイ、ネットワークアドレス変換 (NAT) インスタンス、または VPN 接続を必要とせずに、信頼性の高いスケーラブルな方法で CloudWatch や CloudWatch Synthetics に接続できます。詳細については、『Amazon VPC ユーザーガイド』の「Amazon VPCとは何か」を参照してください。

インターフェイス VPC エンドポイントは AWS PrivateLink を利用しています。これは、Elastic Network Interface とプライベート IP アドレスを使用して AWS のサービス間のプライベート通信を可能にする AWS のテクノロジーです。詳細については、ブログ記事の新規 – AWS サービス用の AWS PrivateLink を参照してください。

以下の手順は、Amazon VPC のユーザー向けです。詳細については、Amazon VPC ユーザーガイドの開始方法を参照してください。

CloudWatch VPC エンドポイント

現在、CloudWatch は以下の AWS リージョンで VPC エンドポイントをサポートしています。

- 米国東部 (オハイオ)
- 米国東部 (バージニア北部)
- 米国西部 (北カリフォルニア)
- 米国西部 (オレゴン)
- アジアパシフィック (香港)
- アジアパシフィック (ムンバイ)
- アジアパシフィック (ソウル)
- アジアパシフィック (シンガポール)
- アジアパシフィック (シドニー)
- アジアパシフィック (東京)
- カナダ (中部)
- 欧州 (フランクフルト)
- ヨーロッパ (アイルランド)
- 欧州 (ロンドン)
- 欧州 (パリ)
- 中東 (アラブ首長国連邦)
- 南米 (サンパウロ)
- AWS GovCloud (米国東部)

- AWS GovCloud (米国西部)

CloudWatch 用の VPC エンドポイントの作成

VPC で CloudWatch の使用を開始するには、CloudWatch のインターフェイス VPC エンドポイントを作成します。サービス名として com.amazonaws.*region*.monitoring を選択します。詳細については、Amazon VPC ユーザーガイドのインターフェイスエンドポイントの作成を参照してください。

CloudWatch の設定を変更する必要はありません。CloudWatch は、パブリックエンドポイントまたはプライベートインターフェイス VPC エンドポイントのうち使用中のいずれかを使用して、他の AWS サービスを呼び出します。例えば、CloudWatch のインターフェイス VPC エンドポイントを作成し、VPC にあるリソースから CloudWatch に流れているメトリクスが既にある場合、それらのメトリクスはデフォルトでインターフェイス VPC エンドポイントを通じて流れ始めます。

CloudWatch の VPC エンドポイントへのアクセスの制御

VPC エンドポイントポリシーは、エンドポイントの作成時または変更時にエンドポイントにアタッチする IAM リソースポリシーです。評価項目の作成時にポリシーを加えない場合、サービスへの数多くのアクセスを許可する初期設定のポリシーがAmazon VPCによって自動的に接続されます。エンドポイントポリシーは、 ユーザーポリシーやサービス固有のポリシーを上書き、または置き換えません。これは、評価項目から指定されたサービスへのアクセスを制御するための別のポリシーです。

評価項目のポリシーは、JSON形式で記載する必要があります。

詳細については、「Amazon VPCユーザーガイド」の「VPC評価項目によるサービスのアクセス制御」を参照してください。

CloudWatch のエンドポイントポリシーの例を次に示します。このポリシーは、VPC を介して CloudWatch に接続するユーザーに対して、CloudWatch にメトリクスデータを送信することを許可し、他の CloudWatch アクションを実行することを禁止します。

```
{
  "Statement": [
    {
      "Sid": "PutOnly",
      "Principal": "*",
      "Action": [
        "cloudwatch:PutMetricData"
      ],
```

```
      "Effect": "Allow",
      "Resource": "*"
    }
  ]
}
```

CloudWatch の VPC エンドポイントポリシーを編集するには

1. Amazon VPCコンソール(https://console.aws.amazon.com/vpc/)を開きます。
2. ナビゲーションペインで、[Endpoints] (エンドポイント) を選択します。
3. CloudWatch のエンドポイントをまだ作成していない場合は、[Create Endpoint (エンドポイントの作成)] を選択します。[com.amazonaws.***region***.monitoring] を選択し、[エンドポイントの作成] を選択します。
4. [com.amazonaws.***region***.monitoring] エンドポイントを選択し、[ポリシー] タブを選択します。
5. [ポリシーの編集] を選択し、変更を行います。

CloudWatch Synthetics の VPC エンドポイント

現在、CloudWatch Synthetics は次の AWS リージョンで VPC エンドポイントをサポートしています。

- 米国東部 (オハイオ)
- 米国東部 (バージニア北部)
- 米国西部 (北カリフォルニア)
- 米国西部 (オレゴン)
- アジアパシフィック (香港)
- アジアパシフィック (ムンバイ)
- アジアパシフィック (ソウル)
- アジアパシフィック (シンガポール)
- アジアパシフィック (シドニー)
- アジアパシフィック (東京)
- カナダ (中部)
- 欧州 (フランクフルト)
- ヨーロッパ (アイルランド)

- 欧州 (ロンドン)
- 欧州 (パリ)
- 南米 (サンパウロ)

CloudWatch Synthetics の VPC エンドポイントの作成

VPC で CloudWatch Synthetics の使用を開始するには、CloudWatch Synthetics 用のインターフェイス VPC エンドポイントを作成します。サービス名として `com.amazonaws.region.synthetics` を選択します。詳細については、Amazon VPC ユーザーガイドのインターフェイスエンドポイントの作成を参照してください。

CloudWatch Synthetics の設定を変更する必要はありません。CloudWatch Synthetics は、パブリックエンドポイントまたはプライベートインターフェイス VPC エンドポイントのうち、いずれか使用中のエンドポイントを使用して、他の AWS サービスと通信します。例えば、CloudWatch Synthetics 用のインターフェイス VPC エンドポイントを作成する場合、Amazon S3 用のインターフェイスエンドポイントが既にあると、CloudWatch Synthetics はデフォルトでインターフェイス VPC エンドポイントを介して Amazon S3 との通信を開始します。

CloudWatch Synthetics の VPC エンドポイントへのアクセスの制御

VPC エンドポイントポリシーは、エンドポイントの作成時または変更時にエンドポイントにアタッチする IAM リソースポリシーです。エンドポイントの作成時にポリシーをアタッチしない場合、サービスへのフルアクセスを許可するデフォルトのポリシーがアタッチされます。エンドポイントポリシーは、 ユーザーポリシーやサービス固有のポリシーを上書き、または置き換えません。これは、エンドポイントから指定されたサービスへのアクセスを制御するための別のポリシーです。

エンドポイントポリシーは、VPC によって非公開で管理される Canary に影響します。プライベートサブネットで実行される Canary には必要ありません。

エンドポイントのポリシーは、JSON 形式で記述される必要があります。

詳細については、「Amazon VPCユーザーガイド」の「VPC評価項目によるサービスのアクセス制御」を参照してください。

CloudWatch Synthetics のエンドポイントポリシーの例を次に示します。このポリシーを使用すると、VPC を介して CloudWatch Synthetics に接続するユーザーは、Canary とその実行に関する情報は表示できますが、Canary を作成、変更、または削除することはできません。

```
{
```

```
    "Statement": [
        {
            "Action": [
                "synthetics:DescribeCanaries",
                "synthetics:GetCanaryRuns"
            ],
            "Effect": "Allow",
            "Resource": "*",
            "Principal": "*"
        }
    ]
}
```

CloudWatch Synthetics の VPC エンドポイントポリシーを編集するには

1. Amazon VPCコンソール(https://console.aws.amazon.com/vpc/)を開きます。
2. ナビゲーションペインで、[Endpoints] (エンドポイント) を選択します。
3. CloudWatch Synthetics のエンドポイントをまだ作成していない場合は、[エンドポイントの作成] を選択します。[com.amazonaws.***region***.synthetics] を選択し、[エンドポイントの作成] を選択します。
4. [com.amazonaws.***region***.synthetics] エンドポイントを選択し、[ポリシー] タブを選択します。
5. [ポリシーの編集] を選択し、変更を行います。

Synthetics Canary のセキュリティ上の考慮事項

次のセクションでは、Synthetics で Canary を作成および実行するときに考慮すべきセキュリティ上の問題について説明します。

セキュリティで保護された接続を使用

Canary コードと Canary テストの実行結果には機密情報が含まれている可能性があるため、Canary は暗号化されていない接続を介してエンドポイントに接続しないでください。`https://` で始まる接続など、常に暗号化された接続を使用してください。

Canary の命名に関する考慮事項

Canary の Amazon リソースネーム (ARN) は、CloudWatch Synthetics ラッパーライブラリの一部として含まれている Puppeteer 主導の Chromium ブラウザからのアウトバウンド呼び出しの一部とし

て、ユーザーエージェントヘッダーに含まれます。これにより、CloudWatch Synthetics Canary トラフィックを特定し、呼び出している Canary に関連付けることができます。

Canary ARN には Canary 名が含まれます。専有情報を明らかにしない Canary の名前を選択してください。

また、Canary は、管理するウェブサイトとエンドポイントにのみ向けるようにしてください。

Canary コードに含まれるシークレットと機密情報

zip ファイルを使用して Canary コードを Canary に直接渡すと、スクリプトの内容を AWS CloudTrail ログで確認することができます。

Canary スクリプトに機密情報やシークレット (アクセスキーやデータベース認証情報など) がある場合は、Canary コードを zip ファイルで渡すのではなく、スクリプトをバージョン管理されたオブジェクトとして Amazon S3 に保存し、Amazon S3 の場所を Canary に渡すことを強くお勧めします。

Canary スクリプトを渡すために zip ファイルを使用する場合は、Canary ソースコードにシークレットや機密情報を含めないことを強くお勧めします。AWS Secrets Manager を使用してシークレットを安全に保つ方法の詳細については、「AWS Secrets Manager の概要」を参照してください。

許可に関する考慮事項

CloudWatch Synthetics によって作成または使用されるリソースへのアクセスを制限することをお勧めします。Canary がテストランの結果とログやスクリーンショットなどのその他のアーティファクトを保存する Amazon S3 バケットには、厳しいアクセス許可を使用します。

同様に、Canary ソースコードが保存されている場所に対するアクセス許可を厳しく保ち、Canary で使用されている Lambda レイヤーや Lambda 関数をユーザーが誤ってまたは悪意を持って削除しないようにします。

意図した Canary コードを確実に実行できるように、Canary コードが保存されている Amazon S3 バケットでオブジェクトバージョニングを使用できます。次に、このコードを Canary として実行するように指定すると、次の例のように、`versionId` オブジェクトをパスの一部として含めることができます。

```
https://bucket.s3.amazonaws.com/path/object.zip?versionId=version-id
https://s3.amazonaws.com/bucket/path/object.zip?versionId=version-id
```

```
https://bucket.s3-region.amazonaws.com/path/object.zip?versionId=version-id
```

スタックトレースと例外メッセージ

デフォルトでは、CloudWatch Synthetics Canary は、スクリプトがカスタムであるか、設計図からのものであるかに関係なく、Canary スクリプトによってスローされた例外をキャプチャします。CloudWatch Synthetics は、例外メッセージとスタックトレースの両方を 3 つの場所に記録します。

- テストランを記述するときにデバッグを高速化する CloudWatch Synthetics サービス
- Lambda 関数が作成した設定に従って CloudWatch Logs に
- Synthetics ログファイル。これは、Canary の `resultsLocation` に設定した値で指定された Amazon S3 ロケーションにアップロードされるプレーンテキストファイルです。

送信および保存する情報を少なくする場合は、CloudWatch Synthetics ラッパーライブラリに戻る前に例外をキャプチャできます。

また、エラーにリクエスト URL を設定することもできます。CloudWatch Synthetics は、スクリプトによってスローされたエラーの URL をスキャンし、restrictedUrlParameters 設定に基づいて、それらから制限された URL パラメータを編集します。スクリプトにエラーメッセージを記録している場合は、getSanitizedErrorMessage を使用して、ロギング前に URL を編集することができます。

IAM ロールの範囲を絞り込む

潜在的に悪意のある URL またはエンドポイントにアクセスするように Canary を設定しないことをお勧めします。信頼できないウェブサイトやエンドポイントに Canary をポイントすると、Lambda 関数コードが悪意のあるユーザーのスクリプトに公開される可能性があります。悪意のあるウェブサイトが Chromium から抜け出す可能性があると仮定すると、インターネットブラウザを使用して接続した場合と同様の方法で Lambda コードにアクセスできる可能性があります。

スコープダウン許可を持つ IAM 実行ロールを使用して Lambda 関数を実行します。このようにして、Lambda 関数が悪意のあるスクリプトによって侵害された場合、Canary の AWS アカウントとして実行するときに実行できるアクションに制限されます。

CloudWatch コンソールを使用して Canary を作成すると、そのコンソールはスコープダウン IAM 実行ロールで作成されます。

機密データのリダクション

CloudWatch Synthetics は、URL、ステータスコード、エラーの理由 (存在する場合)、およびリクエストとレスポンスのヘッダーと本文をキャプチャします。これにより、Canary ユーザは、Canary を理解、モニタリング、デバッグすることができます。

次のセクションで説明する構成は、Canary 実行の任意の時点で設定できます。また、異なる Synthetics ステップに異なる設定を適用することもできます。

リクエスト URL

デフォルトでは、CloudWatch Synthetics ログは URL、ステータスコード、および Canary ログ内の各 URL のステータス理由をリクエストします。リクエスト URL は、Canary 実行レポート、HAR ファイルなどにも表示できます。リクエスト URL には、アクセストークンやパスワードなどの機密性の高いクエリパラメータが含まれている場合があります。CloudWatch Synthetics によって機密情報が記録されないように編集することができます。

機密情報を編集するには、設定プロパティ restrictedUrlParameters を設定します。詳細については、「SyntheticsConfiguration クラス」を参照してください。これにより、CloudWatch Synthetics は URL パラメータ (パスおよびクエリパラメータ値を含む) を ログを記録する前に restrictedUrlParameters に基づいて編集します。スクリプトに URL を記録している場合は、getSanitizedUrl(url, stepConfig = null) を使用して、ロギング前に URL を編集することができます。詳細については、「SyntheticsLogHelper クラス」を参照してください。

ヘッダー

デフォルトでは、CloudWatch Synthetics はリクエスト/レスポンスヘッダーを記録しません。UI Canary の場合、これはランタイムバージョン `syn-nodejs-puppeteer-3.2` 以降を使用する Canary のデフォルトの動作です。

ヘッダーに機密情報が含まれていない場合は、HAR ファイルおよび HTTP レポートでヘッダーを有効にするには、includeRequestHeaders および includeResponseHeaders プロパティを `true` に設定します。すべてのヘッダーを有効にできますが、機密性の高いヘッダーキーの値を制限することを選択できます。例えば、Canary によって生成されたアーティファクトから `Authorization` ヘッダーだけを編集するように選択できます。

リクエストとレスポンスの本文

デフォルトでは、CloudWatch Synthetics はリクエスト/レスポンス本文を Canary ログまたはレポートに記録しません。この情報は、API Canary で特に有益です。Synthetics は、すべてのHTTPリクエ

ストをキャプチャし、ヘッダー、リクエスト、レスポンスの本文を表示できます。詳細については、「executeHttpStep(stepName, requestOptions, [callback], [stepConfig])」を参照してください。リクエスト/レスポンス本文を有効にするかどうかは、includeRequestBody および includeResponseBody プロパティを `true` に設定することで選択できます。

AWS CloudTrail を使用した Amazon CloudWatch API コールのログ記録

Amazon CloudWatch および CloudWatch Synthetics は、ユーザー、ロール、または AWS のサービスによって実行されたアクションを記録するサービスである AWS CloudTrail と統合されています。CloudTrail は、AWS アカウントによって行われた API コール、またはそのアカウントの代わりに行われた API コールをキャプチャします。キャプチャされるコールには、コンソールからの呼び出しと、API オペレーションへのコード呼び出しが含まれます。

証跡を作成する場合は、CloudWatch のイベントなど、S3 バケットへの CloudTrail イベントの継続的な配信を有効にすることができます。追跡を設定しない場合でも、CloudTrail コンソールの [Event history] (イベント履歴) で最新のイベントを表示できます。CloudTrail で収集された情報を使用して、CloudWatch に対するリクエスト、リクエスト元の IP アドレス、リクエスト者、リクエスト日時などの詳細を確認できます。

設定や有効化の方法など、CloudTrail の詳細については、「AWS CloudTrail ユーザーガイド」を参照してください。

各イベントまたはログエントリには、リクエストの生成者に関する情報が含まれます。アイデンティティ情報は、以下を判別するのに役立ちます。

- リクエストが、ルート認証情報と AWS Identity and Access Management (IAM) ユーザー認証情報のどちらを使用して送信されたか。
- リクエストがロールまたはフェデレーティッドユーザーのテンポラリなセキュリティ認証情報を使用して行われたかどうか。
- リクエストが、別の AWS のサービスによって送信されたかどうか。

詳細については、「CloudTrail userIdentity エレメント」を参照してください。

CloudWatch および CloudWatch Synthetics のイベントなど、AWS アカウントのイベントの継続的な記録については、証跡を作成します。[trail] (証跡)より、CloudTrail はログファイルを S3 バケットに配信できます。デフォルトでは、コンソールで証跡を作成するときに、証跡がすべての AWS リージョンに適用されます。追跡では、AWS パーティションのすべてのリージョンからのイベントがログに記録され、指定した S3 バケットにログファイルが配信されます。その他の AWS のサービスを設定して、CloudTrail ログで収集されたデータをより詳細に分析し、それに基づく対応を行うことができます。詳細については、次を参照してください。

- 追跡を作成するための概要
- CloudTrail のサポート対象サービスと統合
- Amazon SNS の CloudTrail の通知の設定
- 「複数のリージョンから CloudTrail ログファイルを受け取る」および「複数のアカウントから CloudTrail ログファイルを受け取る」

トピック

- CloudTrail の CloudWatch 情報
- CloudTrail の CloudWatch Internet Monitor
- CloudTrail の CloudWatch Synthetics 情報

CloudTrail の CloudWatch 情報

CloudWatch は、CloudTrail ログファイルのイベントとして以下のアクションを記録します。

- DeleteAlarms
- DeleteAnomalyDetector
- DeleteDashboards
- DescribeAlarmHistory
- DescribeAlarms
- DescribeAlarmsForMetric
- DescribeAnomalyDetectors
- DisableAlarmActions
- EnableAlarmActions
- GetDashboard
- ListDashboards
- PutAnomalyDetector
- PutDashboard
- PutMetricAlarm
- SetAlarmState

例: CloudWatch ログファイルのエントリ

次の例は、PutMetricAlarm アクションを示す CloudTrail ログエントリです。


```
{
    "Records": [{
        "eventVersion": "1.01",
        "userIdentity": {
            "type": "Root",
            "principalId": "EX_PRINCIPAL_ID",
            "arn": "arn:aws:iam::123456789012:root",
            "accountId": "123456789012",
            "accessKeyId": "EXAMPLE_KEY_ID"
        },
        "eventTime": "2014-03-23T21:50:34Z",
        "eventSource": "monitoring.amazonaws.com",
        "eventName": "PutMetricAlarm",
        "awsRegion": "us-east-1",
        "sourceIPAddress": "127.0.0.1",
        "userAgent": "aws-sdk-ruby2/2.0.0.rc4 ruby/1.9.3 x86_64-linux Seahorse/0.1.0",
        "requestParameters": {
            "threshold": 50.0,
            "period": 60,
            "metricName": "CloudTrail Test",
            "evaluationPeriods": 3,
            "comparisonOperator": "GreaterThanThreshold",
            "namespace": "AWS/CloudWatch",
            "alarmName": "CloudTrail Test Alarm",
            "statistic": "Sum"
        },
        "responseElements": null,
        "requestID": "29184022-b2d5-11e3-a63d-9b463e6d0ff0",
        "eventID": "b096d5b7-dcf2-4399-998b-5a53eca76a27"
    },
    ..additional entries
  ]
  }
```


以下のログファイルエントリは、ユーザーが CloudWatch Events PutRule アクションを呼び出したことを示します。


```
{
         "eventVersion":"1.03",
```

```
         "userIdentity":{
            "type":"Root",
            "principalId":"123456789012",
            "arn":"arn:aws:iam::123456789012:root",
            "accountId":"123456789012",
            "accessKeyId":"AKIAIOSFODNN7EXAMPLE",
            "sessionContext":{
               "attributes":{
                  "mfaAuthenticated":"false",
                  "creationDate":"2015-11-17T23:56:15Z"
               }
            }
         },
         "eventTime":"2015-11-18T00:11:28Z",
         "eventSource":"events.amazonaws.com",
         "eventName":"PutRule",
         "awsRegion":"us-east-1",
         "sourceIPAddress":"AWS Internal",
         "userAgent":"AWS CloudWatch Console",
         "requestParameters":{
            "description":"",
            "name":"cttest2",
            "state":"ENABLED",
            "eventPattern":"{\"source\":[\"aws.ec2\"],\"detail-type\":[\"EC2 Instance
 State-change Notification\"]}",
            "scheduleExpression":""
         },
         "responseElements":{
            "ruleArn":"arn:aws:events:us-east-1:123456789012:rule/cttest2"
         },
         "requestID":"e9caf887-8d88-11e5-a331-3332aa445952",
         "eventID":"49d14f36-6450-44a5-a501-b0fdcdfaeb98",
         "eventType":"AwsApiCall",
         "apiVersion":"2015-10-07",
         "recipientAccountId":"123456789012"
}
```

以下のログファイルエントリは、ユーザーが CloudWatch Logs `CreateExportTask` アクションを呼び出したことを示します。

```
{
        "eventVersion": "1.03",
        "userIdentity": {
```

```
            "type": "IAMUser",
            "principalId": "EX_PRINCIPAL_ID",
            "arn": "arn:aws:iam::123456789012:user/someuser",
            "accountId": "123456789012",
            "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
            "userName": "someuser"
        },
        "eventTime": "2016-02-08T06:35:14Z",
        "eventSource": "logs.amazonaws.com",
        "eventName": "CreateExportTask",
        "awsRegion": "us-east-1",
        "sourceIPAddress": "127.0.0.1",
        "userAgent": "aws-sdk-ruby2/2.0.0.rc4 ruby/1.9.3 x86_64-linux Seahorse/0.1.0",
        "requestParameters": {
            "destination": "yourdestination",
            "logGroupName": "yourloggroup",
            "to": 123456789012,
            "from": 0,
            "taskName": "yourtask"
        },
        "responseElements": {
            "taskId": "15e5e534-9548-44ab-a221-64d9d2b27b9b"
        },
        "requestID": "1cd74c1c-ce2e-12e6-99a9-8dbb26bd06c9",
        "eventID": "fd072859-bd7c-4865-9e76-8e364e89307c",
        "eventType": "AwsApiCall",
        "apiVersion": "20140328",
        "recipientAccountId": "123456789012"
}
```


CloudTrail の CloudWatch Internet Monitor

CloudWatch Internet Monitor では、CloudTrail ログファイルのイベントとして、次のアクションのログ記録がサポートされています (リリースでは、API リファレンスガイドへのリンクが追加されます)。

例: CloudWatch Internet Monitor ログファイルエントリ

次の例は、`ListMonitors` アクションを示す CloudTrail Internet Monitor ログエントリです。


```
{
        "eventVersion": "1.08",
```

```
        "userIdentity": {
            "type": "AssumedRole",
            "principalId": "EX_PRINCIPAL_ID",
            "arn": "arn:aws:iam::000000000000:assumed-role/role_name",
            "accountId":"123456789012",
            "accessKeyId":"AKIAIOSFODNN7EXAMPLE",
            "sessionContext": {
                "sessionIssuer": {
                "type": "Role",
                "principalId": "EX_PRINCIPAL_ID",
                "arn": "arn:aws:iam::000000000000:role/Admin",
                "accountId":"123456789012",
                "userName": "SAMPLE_NAME"
                },
                "webIdFederationData": {},
                "attributes": {
                    "creationDate": "2022-10-11T17:25:41Z",
                    "mfaAuthenticated": "false"
                }
            }
        },
        "eventTime": "2022-10-11T17:30:18Z",
        "eventSource": "internetmonitor.amazonaws.com",
        "eventName": "ListMonitors",
        "awsRegion": "us-east-2",
        "sourceIPAddress": "192.0.2.0",
        "userAgent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7)",
        "requestParameters": null,
        "responseElements": null,
        "requestID": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "eventID": "a1b2c3d4-5678-90ab-cdef-EXAMPLEbbbbb",
        "readOnly": true,
        "eventType": "AwsApiCall",
        "managementEvent": true,
        "recipientAccountId": "111122223333",
        "eventCategory": "Management"
    }
```


次の例は、`CreateMonitor` アクションを示す CloudTrail Internet Monitor ログエントリです。


```
{
        "eventVersion": "1.08",
        "userIdentity": {
```

```
            "type": "AssumedRole",
            "principalId": "EX_PRINCIPAL_ID",
            "arn": "arn:aws:iam::000000000000:assumed-role/role_name",
            "accountId":"123456789012",
            "accessKeyId":"AKIAIOSFODNN7EXAMPLE",
            "sessionContext": {
                "sessionIssuer": {
                "type": "Role",
                "principalId": "EX_PRINCIPAL_ID",
                "arn": "arn:aws:iam::000000000000:role/Admin",
                "accountId":"123456789012",
                "userName": "SAMPLE_NAME"
                },
                "webIdFederationData": {},
                "attributes": {
                    "creationDate": "2022-10-11T17:25:41Z",
                    "mfaAuthenticated": "false"
                }
            }
        },
        "eventTime": "2022-10-11T17:30:08Z",
        "eventSource": "internetmonitor.amazonaws.com",
        "eventName": "CreateMonitor",
        "awsRegion": "us-east-2",
        "sourceIPAddress": "192.0.2.0",
        "userAgent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7)",
        "requestParameters": {
            "MonitorName": "TestMonitor",
            "Resources": ["arn:aws:ec2:us-east-2:444455556666:vpc/vpc-febc0b95"],
            "ClientToken": "a1b2c3d4-5678-90ab-cdef-EXAMPLE33333"
        },
        "responseElements": {
            "Arn": "arn:aws:internetmonitor:us-east-2:444455556666:monitor/ct-
onboarding-test",
            "Status": "PENDING"
        },
        "requestID": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "eventID": "a1b2c3d4-5678-90ab-cdef-EXAMPLEbbbbb",
        "readOnly": false,
        "eventType": "AwsApiCall",
        "managementEvent": true,
        "recipientAccountId": "111122223333",
        "eventCategory": "Management"
```

```
    }
```

CloudTrail の CloudWatch Synthetics 情報

CloudWatch Synthetics は、CloudTrail ログファイルのイベントとして以下のアクションを記録します。

- CreateCanary
- DeleteCanary
- DescribeCanaries
- DescribeCanariesLastRun
- DescribeRuntimeVersions
- GetCanary
- GetCanaryRuns
- ListTagsForResource
- StartCanary
- StopCanary
- TagResource
- UntagResource
- UpdateCanary

例: CloudWatch Synthetics のログファイルのエントリ

次の例は、`DescribeCanaries` アクションを示す CloudTrail Synthetics ログエントリです。

```
{
    "eventVersion": "1.05",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "EX_PRINCIPAL_ID",
        "arn": "arn:aws:iam::123456789012:assumed-role/role_name",
        "accountId":"123456789012",
        "accessKeyId":"AKIAIOSFODNN7EXAMPLE",
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
```

```
                "principalId": "EX_PRINCIPAL_ID",
                "arn": "arn:aws:iam::111222333444:role/Administrator",
                "accountId":"123456789012",
                "userName": "SAMPLE_NAME"
            },
            "webIdFederationData": {},
            "attributes": {
                "mfaAuthenticated": "false",
                "creationDate": "2020-04-08T21:43:24Z"
            }
        }
    },
    "eventTime": "2020-04-08T23:06:47Z",
    "eventSource": "synthetics.amazonaws.com",
    "eventName": "DescribeCanaries",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "127.0.0.1",
    "userAgent": "aws-internal/3 aws-sdk-java/1.11.590
 Linux/4.9.184-0.1.ac.235.83.329.metal1.x86_64 OpenJDK_64-Bit_Server_VM/25.212-b03
 java/1.8.0_212 vendor/Oracle_Corporation",
    "requestParameters": null,
    "responseElements": null,
    "requestID": "201ed5f3-15db-4f87-94a4-123456789",
    "eventID": "73ddbd81-3dd0-4ada-b246-123456789",
    "readOnly": true,
    "eventType": "AwsApiCall",
    "recipientAccountId": "111122223333"
}
```

次の例は、UpdateCanary アクションを示す CloudTrail Synthetics ログエントリです。

```
{
    "eventVersion": "1.05",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "EX_PRINCIPAL_ID",
        "arn": "arn:aws:iam::123456789012:assumed-role/role_name",
        "accountId":"123456789012",
        "accessKeyId":"AKIAIOSFODNN7EXAMPLE",
        "sessionContext": {
            "sessionIssuer": {
               "type": "Role",
                "principalId": "EX_PRINCIPAL_ID",
```

```
                "arn": "arn:aws:iam::111222333444:role/Administrator",
       "accountId":"123456789012",
                "userName": "SAMPLE_NAME"
            },
            "webIdFederationData": {},
            "attributes": {
                "mfaAuthenticated": "false",
                "creationDation": "2020-04-08T21:43:24Z"
            }
        }
    },
    "eventTime": "2020-04-08T23:06:47Z",
    "eventSource": "synthetics.amazonaws.com",
    "eventName": "UpdateCanary",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "192.0.2.0",
    "userAgent": "aws-internal/3 aws-sdk-java/1.11.590
 Linux/4.9.184-0.1.ac.235.83.329.metal1.x86_64 OpenJDK_64-Bit_Server_VM/25.212-b03
 java/1.8.0_212 vendor/Oracle_Corporation",
    "requestParameters": {
        "Schedule": {
            "Expression": "rate(1 minute)"
        },
        "name": "sample_canary_name",
        "Code": {
            "Handler": "myOwnScript.handler",
            "ZipFile": "SAMPLE_ZIP_FILE"
        }
    },
    "responseElements": null,
    "requestID": "fe4759b0-0849-4e0e-be71-1234567890",
    "eventID": "9dc60c83-c3c8-4fa5-bd02-1234567890",
    "readOnly": false,
    "eventType": "AwsApiCall",
    "recipientAccountId": "111122223333"
}
```


次の例は、GetCanaryRuns アクションを示す CloudTrail Synthetics ログエントリです。


```
{
    "eventVersion": "1.05",
    "userIdentity": {
        "type": "AssumedRole",
```

```
        "principalId": "EX_PRINCIPAL_ID",
        "arn": "arn:aws:iam::123456789012:assumed-role/role_name",
        "accountId":"123456789012",
        "accessKeyId":"AKIAIOSFODNN7EXAMPLE",
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
                "principalId": "EX_PRINCIPAL_ID",
                "arn": "arn:aws:iam::111222333444:role/Administrator",
       "accountId":"123456789012",
                "userName": "SAMPLE_NAME"
            },
            "webIdFederationData": {},
            "attributes": {
                "mfaAuthenticated": "false",
                "creationDate": "2020-04-08T21:43:24Z"
            }
        }
    },
    "eventTime": "2020-04-08T23:06:30Z",
    "eventSource": "synthetics.amazonaws.com",
    "eventName": "GetCanaryRuns",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "127.0.0.1",
    "userAgent": "aws-internal/3 aws-sdk-java/1.11.590
 Linux/4.9.184-0.1.ac.235.83.329.metal1.x86_64 OpenJDK_64-Bit_Server_VM/25.212-b03
 java/1.8.0_212 vendor/Oracle_Corporation",
    "requestParameters": {
        "Filter": "TIME_RANGE",
        "name": "sample_canary_name",
        "FilterValues": [
            "2020-04-08T23:00:00.000Z",
            "2020-04-08T23:10:00.000Z"
        ]
    },
    "responseElements": null,
    "requestID": "2f56318c-cfbd-4b60-9d93-1234567890",
    "eventID": "52723fd9-4a54-478c-ac55-1234567890",
    "readOnly": true,
    "eventType": "AwsApiCall",
    "recipientAccountId": "111122223333"
}
```

Amazon CloudWatch リソースにタグを付ける

タグは、ユーザーまたは AWS が AWS リソースに割り当てるカスタム属性ラベルです。各 タグは 2 つの部分で構成されます:

- タグキー (例: `CostCenter`、`Environment`、または `Project`)。タグキーでは、大文字と小文字が区別されます。
- タグ値として知られるオプションのフィールド (例: `111122223333` または `Production`)。タグ値を省略すると、空の文字列を使用した場合と同じになります。タグキーと同様に、タグ値は大文字と小文字が区別されます。

タグは、以下のことに役立ちます。

- AWS リソースの特定と整理。多くの AWS のサービスではタグ付けがサポートされるため、さまざまなサービスからリソースに同じタグを割り当てて、リソースの関連を示すことができます。例えば、EC2 インスタンスに割り当てる同じタグを CloudWatch ルールに割り当てることができます。

以下のセクションでは、CloudWatch のタグに関する詳細を示します。

CloudWatch でサポートされるリソース

CloudWatch の次のリソースは、タグ付けをサポートしています。

- アラーム - アラームにタグを付けるには、tag-resource AWS CLI コマンドと TagResource API を使用します。CloudWatch コンソールの [アラーム] の詳細ページを使用して、アラームタグを表示および管理することもできます。
- Canary – Canary にタグを付けるには、CloudWatch コンソールを使用できます。詳細については、「Canary を作成する」を参照してください。
- Contributor Insights ルール - Contributor Insights ルールにタグを付けるには、そのルールの作成時に put-insight-rule AWS CLI コマンドと PutInsightRule API を使用します。既存のルールにタグを追加するには、tag-resource AWS CLI コマンドと TagResource API を使用できます。
- メトリクスストリーム - メトリクスストリームを作成する場合、put-metric-stream AWS CLI コマンドと PutMetricStream API を使用して、メトリクスストリームにタグを付けることができます。既存のメトリクスストリームタグを追加するには、tag-resource AWS CLI コマンドと TagResource API を使用します。

タグの追加と管理の詳細については、「タグの管理」を参照してください。

タグの管理

タグは、リソースの Key および Value プロパティで構成されています。このようなプロパティの値を追加、編集、削除するには、CloudWatch コンソール、AWS CLI、または CloudWatch API を使用できます。タグの使用については、以下を参照してください。

- 「Amazon CloudWatch API リファレンス」の「TagResource」、「UntagResource」、「ListTagsForResource」
- 『Amazon CloudWatch CLI リファレンス』の tag-resource、untag-resource、および list-tags-for-resource
- Resource Groups ユーザーガイドの「タグエディタの使用」

タグの命名規則と使用規則

CloudWatch リソースでのタグの使用には、次の基本的な命名規則と使用規則が適用されます。

- 各リソースには、最大 50 個のタグを設定できます。
- タグキーは、リソースごとにそれぞれ一意である必要があります。また、各タグキーに設定できる値は 1 つのみです。
- タグキーの最大長は UTF-8 で 128 Unicode 文字です。
- タグ値の最大長は UTF-8 で 256 Unicode 文字です。
- 使用できる文字は、UTF-8 対応の文字、数字、スペースと、文字 (. : + = @ _ / -) (ハイフン) です。
- タグのキーと値は大文字と小文字が区別されます。ベストプラクティスとして、タグを大文字にするための戦略を決定し、その戦略をすべてのリソースタイプにわたって一貫して実装します。たとえば、Costcenter、costcenter、CostCenter を使用するかどうかを決定し、すべてのタグに同じ規則を使用します。大文字と小文字の扱いについて、同様のタグに整合性のない規則を使用することは避けてください。
- aws: プレフィックスは AWS の使用に予約されているため、タグで使用することはできません。このプレフィックスが含まれるタグのキーや値を編集または削除することはできません。このプレフィックスを持つタグは、リソースあたりのタグ数の制限にはカウントされません。

Grafana の統合

Grafana バージョン 6.5.0 以降を使用して、コンテキストに応じて CloudWatch コンソールに進み、ワイルドカードを使用してメトリクスの動的なリストをクエリできます。これは、Amazon Elastic Compute Cloud インスタンスやコンテナといった AWS リソースのメトリクスのモニタリングに役立ちます。Auto Scaling イベントの一部として新しいインスタンスが作成されると、自動的にグラフに表示されます。新しいインスタンス ID を追跡する必要はありません。事前構築済みのダッシュボードにより、Amazon EC2、Amazon Elastic Block Store、AWS Lambda リソースのモニタリングを簡単に開始できます。

Grafana バージョン 7.0 以降を使用して、CloudWatch Logs のロググループに対して CloudWatch Logs Insights クエリを実行できます。クエリ結果は、棒グラフ、折れ線グラフ、積み上げグラフ、および表形式で視覚化できます。CloudWatch Logs Insights の詳細については、「CloudWatch Logs Insights を使用したログデータの分析」を参照してください。

開始方法の詳細については、Grafana Labs ドキュメントの「Grafana で AWS CloudWatch を使用する」を参照してください。

クロスアカウントクロスリージョン CloudWatch コンソール

メトリクス、ログ、トレースに対するクロスアカウントのオブザーバビリティと検出を最大限に活用するには、CloudWatch クロスアカウントオブザーバビリティを使用することをお勧めします。詳細については、「CloudWatch のクロスアカウントオブザーバビリティ」を参照してください。

CloudWatch には、クロスアカウントクロスリージョン CloudWatch ダッシュボードも用意されています。この機能により、ダッシュボード、アラーム、メトリクス、自動ダッシュボードをクロスアカウントで確認することが可能になります。ログやトレースをクロスアカウントで確認することはできません。

CloudWatch のクロスアカウントオブザーバビリティも使用している場合、このクロスアカウント CloudWatch ダッシュボードのユースケースの 1 つは、CloudWatch クロスアカウントオブザーバビリティソースアカウントに、別のソースアカウントのメトリクスを確認させることです。

このセクションでは、この後、クロスアカウントクロスリージョンダッシュボードについて説明します。これを使用して、複数の AWS アカウントと複数の AWS リージョンからの CloudWatch データを 1 つのダッシュボードにまとめたダッシュボードを作成できます。別のアカウントにあるメトリックスを監視するアラームを 1 つのアカウントで作成することもできます。

多くの組織では、請求とセキュリティの境界を提供するために、AWS リソースを複数のアカウントにデプロイしています。この場合、モニターリングアカウントとして 1 つまたは複数のアカウントを指定し、これらのアカウントでクロスアカウントダッシュボードを構築することをお勧めします。

クロスアカウント機能は AWS Organizations と統合されているため、クロスアカウントダッシュボードを効率的に構築できます。

クロスリージョン機能

クロスリージョン機能が自動的に組み込まれるようになりました。追加の手順を実行しなくても、単一のアカウントで複数の異なるリージョンからのメトリクスを同じグラフや同じダッシュボードに表示できます。クロスリージョン機能はアラームではサポートされていないため、あるリージョンで作成したアラームで別のリージョンのメトリクスをモニターリングすることはできません。

トピック

- CloudWatch でのクロスアカウント機能の有効化
- (オプション) との統合AWS Organizations
- CloudWatch クロスアカウントセットアップのトラブルシューティング
- クロスアカウント使用後の無効化とクリーンアップ

CloudWatch でのクロスアカウント機能の有効化

CloudWatch コンソールでクロスアカウント機能を設定するには、CloudWatch コンソールを使用して共有アカウントとモニターリングアカウントを設定します。

共有アカウントを設定する

モニターリングアカウントでデータを使用できるようにするには、各アカウントで共有を有効にする必要があります。

これにより、共有先のアカウントでクロスアカウントダッシュボードを表示するすべてのユーザーに対して、当該ユーザーが共有先のアカウントで対応する許可を有している場合は、ステップ 5 で選択した読み取り専用許可が付与されます。

アカウントが他のアカウントと CloudWatch データを共有できるようにするには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで 設定 を選択します。
3. [Share your CloudWatch data] (CloudWatch データを共有) で [Configure] (設定) を選択します。
4. [共有] で [特定のアカウント] を選択し、データを共有するアカウントの ID を入力します。

   ここで指定したアカウントは、アカウントの CloudWatch データを表示できます。すでに認識していて信頼できるアカウントの ID のみを指定します。
5. [アクセス権限] で、次のいずれかのオプションを使用してデータを共有する方法を指定します。

   - CloudWatch のメトリクス、ダッシュボード、アラームへの読み取り専用アクセスを提供する: このオプションを使用すると、モニターリングアカウントで、アカウントの CloudWatch データが入ったウィジェットを含むクロスアカウントダッシュボードを作成できます。
   - CloudWatch 自動ダッシュボードを含めます。このオプションを選択すると、モニターリングアカウントのユーザーは、このアカウントの自動ダッシュボードの情報も表示できます。詳細については、「Amazon CloudWatch の開始方法」を参照してください。
   - X-Ray トレースマップに対する X-Ray 読み取り専用アクセスを含めます。このオプションを選択すると、モニタリングアカウントのユーザーは、このアカウントの X-Ray トレースマップと X-Ray トレース情報も表示できます。詳細については、「Using the X-Ray Trace Map」を参照してください。
   - アカウント内のすべての情報への完全な読み取り専用アクセス: このオプションを使用すると、共有用のアカウントで、アカウントの CloudWatch データが入ったウィジェットを含むクロスアカウントダッシュボードを作成できます。また、これらのアカウントは、アカウントを

より深く調べて、他の AWS サービスのコンソールでアカウントのデータを表示することもできます。

6. [Launch CloudFormation template (CloudFormation の起動テンプレート)] を選択します。

   確認画面で、「**Confirm**」と入力し、[起動テンプレート] を選択します。

7. [I acknowledge...] チェックボックスをオンにして、[スタックの作成] を選択します。

組織全体との共有

上記の手順を完了すると、アカウントが 1 つのアカウントとデータを共有できる IAM ロールが作成されます。組織内のすべてのアカウントとデータを共有する IAM ロールを作成または編集できます。これは、組織内のすべてのアカウントについて認識していて、信頼できる場合にのみ実行してください。

これにより、共有先のアカウントでクロスアカウントダッシュボードを表示するすべてのユーザーに対して、当該ユーザーが共有先のアカウントで対応する許可を有している場合は、前の手順のステップ 5 で示したポリシーにリストされた読み取り専用許可が付与されます。

組織内のすべてのアカウントと CloudWatch アカウントデータを共有するには

1. まだ 1 つの AWS アカウントとデータを共有していない場合は、上記の手順を完了します。
2. AWS Management Console にサインインして、https://console.aws.amazon.com/iam/ で IAM コンソールを開きます。
3. ナビゲーションペインで [Roles] (ロール) を選択します。
4. ロールのリストで、[CloudWatch-CrossAccountSharingRole] を選択します。
5. [信頼関係]、[信頼関係の編集] の順に選択します。

   次のようなポリシーが表示されます。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::123456789012:root"
      },
      "Action": "sts:AssumeRole"
    }
```

```
  ]
}
```

6. ポリシーを次のように変更し、 *org-id* を組織の ID に置き換えます。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "*"
      },
      "Action": "sts:AssumeRole",
      "Condition": {
        "StringEquals": {
          "aws:PrincipalOrgID": "org-id"
        }
      }

    }
  ]
}
```

7. 信頼ポリシーの更新 を選択します。

監視アカウントを設定

クロスアカウント CloudWatch データを表示する場合は、各モニターリングアカウントを有効にします。

次の手順を完了すると、CloudWatch は、モニターリングアカウントで CloudWatch を使用して他のアカウントから共有されるデータにアクセスする、サービスにリンクされたロールを作成します。このサービスリンクロールは、AWSServiceRoleForCloudWatchCrossAccount と呼ばれます。詳細については、「CloudWatch のサービスにリンクされたロールの使用 」を参照してください。

クロスアカウント CloudWatch データをアカウントで表示できるようにするには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで、[Settings] (設定) を選択してから、[Cross-account cross-region] (クロスアカウントクロスリージョン) セクションで、[Configure] (設定) を選択します。

3. [View cross-account cross-region] (クロスアカウントクロスリージョンを表示) セクションで、[Enable] (有効化) を選択し、[Show selector in the console] (コンソールにセレクタを表示する) チェックボックスをオンにして、メトリクススのグラフ化またはアラームの作成時にアカウントセレクターが CloudWatch コンソールに表示されるようにします。
4. [View cross-account cross-region (クロスアカウントクロスリージョンの表示)] で、次のいずれかのオプションを選択します。

   - アカウント ID の入力: このオプションでは、クロスアカウントデータを表示するときにアカウントを切り替えるたびに、手動でアカウント ID を入力するように求められます。
   - AWS Organization アカウントセレクタ。このオプションを使用すると、Organizations とのクロスアカウント統合の完了時に指定したアカウントが表示されます。次回コンソールを使用すると、クロスアカウントデータを表示したとき、CloudWatch にこれらのアカウントのドロップダウンリストが表示され、アカウントを選択できます。

     これを行うには、最初に組織のマネジメントアカウントを使用して、CloudWatch が組織内のアカウントのリストを参照できるようにする必要があります。詳細については、「(オプション) との統合AWS Organizations」を参照してください。
   - カスタムアカウントセレクタ: このオプションでは、アカウント ID のリストを入力するように求められます。次回コンソールを使用すると、クロスアカウントデータを表示したとき、CloudWatch にこれらのアカウントのドロップダウンリストが表示され、アカウントを選択できます。

     また、表示するアカウントを選択するときに識別しやすくするため、各アカウントのラベルを入力することもできます。

     ユーザーがここで行うアカウントセレクタ設定は、そのユーザーに対してのみ保持され、モニターリングするアカウント内の他のすべてのユーザーに対しては保持されません。
5. [Enable (有効化)] を選択します。

この設定を完了すると、クロスアカウントダッシュボードを作成できます。詳細については、「クロスアカウントクロスリージョンダッシュボード」を参照してください。

(オプション) との統合AWS Organizations

クロスアカウント機能を AWS Organizations と統合する場合は、モニターリングアカウントとして使用できる組織内のすべてのアカウントのリストを作成する必要があります。

クロスアカウント CloudWatch 機能を有効にして、組織内のすべてのアカウントのリストにアクセスするには

1. 組織の管理アカウントにサインインします。
2. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
3. ナビゲーションペインで、[設定] を選択し、[設定] を選択します。
4. [Grant permission to view the list of accounts in the organization (組織内のアカウントのリストを表示するアクセス権限を付与)] で、[Specific accounts (特定のアカウント)] を選択すると、アカウント ID のリストの入力を求められます。組織内のアカウントのリストは、ここで指定したアカウントとのみ共有されます。
5. [Share organization account (組織アカウントリストの共有)] を選択します。
6. [Launch CloudFormation template (CloudFormation の起動テンプレート)] を選択します。

   確認画面で、「**`Confirm`**」と入力し、[起動テンプレート] を選択します。

CloudWatch クロスアカウントセットアップのトラブルシューティング

このセクションでは、CloudWatch でのクロスアカウントコンソールのデプロイに関するトラブルシューティングのヒントを示します。

クロスアカウントデータの表示時にアクセス拒否エラーが発生する

以下を確認してください。

- モニターリングアカウントには、AWSServiceRoleForCloudWatchCrossAccount という名前のロールが必要です。ない場合は、このロールを作成する必要があります。詳細については、「Set Up a Monitoring Account」を参照してください。
- 各共有アカウントには、CloudWatch-CrossAccountSharingRole という名前のロールが必要です。ない場合は、このロールを作成する必要があります。詳細については、「Set Up A Sharing Account」を参照してください。
- 共有ロールは、モニターリングアカウントを信頼する必要があります。

CloudWatch クロスアカウントコンソール用にロールが正しく設定されていることを確認するには

1. AWS Management Console にサインインして、https://console.aws.amazon.com/iam/ で IAM コンソールを開きます。
2. ナビゲーションペインで [Roles] (ロール) を選択します。
3. ロールのリストで、必要なロールが存在することを確認します。共有アカウントで、CloudWatch-CrossAccountSharingRole を探します。モニターリングアカウントで、AWSServiceRoleForCloudWatchCrossAccount を探します。
4. 共有アカウントを使用していて、CloudWatch-CrossAccountSharingRole がすでに存在する場合は、CloudWatch-CrossAccountSharingRole を選択します。
5. [信頼関係]、[信頼関係の編集] の順に選択します。
6. ポリシーに、モニターリングアカウントのアカウント ID、またはモニターリングアカウントを含む組織の組織 ID のいずれかが表示されていることを確認します。

コンソールにアカウントドロップダウンが表示されない

まず、前のトラブルシューティングのセクションで説明したように、正しい IAM ロールが作成されていることを確認します。正しく設定されている場合は、「Enable Your Account to View Cross-Account Data」で説明されているように、このアカウントでクロスアカウントデータを表示できるようにしてください。

クロスアカウント使用後の無効化とクリーンアップ

CloudWatch のクロスアカウント機能を無効にするには、以下のステップに従います。

ステップ 1: クロスアカウントスタックまたはロールを削除する

最善の方法は、クロスアカウント機能を有効にするために使用された AWS CloudFormation スタックを削除することです。

- 各共有アカウントで、CloudWatch-CrossAccountSharingRole スタックを削除します。
- AWS Organizations を使用して、組織内のすべてのアカウントでクロスアカウント機能を有効にしていた場合は、組織の管理アカウントにある CloudWatch-CrossAccountListAccountsRole スタックを削除します。

クロスアカウント機能を有効にするために AWS CloudFormation スタックを使用していない場合は、以下の操作を行います。

- 各共有アカウントで、CloudWatch-CrossAccountSharingRole IAM ロールを削除します。
- AWS Organizations を使用して、組織内のすべてのアカウントでクロスアカウント機能を有効にしていた場合は、組織の管理アカウントで CloudWatch-CrossAccountSharing-ListAccountsRole IAM ロールを削除します 。

ステップ 2: サービスにリンクされたロールを削除する

モニターリングアカウントで、AWSServiceRoleForCloudWatchCrossAccount サービスにリンクされた IAM ロールを削除します。

CloudWatch サービスのクォータ

CloudWatch には、メトリクス、アラーム、API リクエスト、E メール通知に関する次のクォータがあります。

> **Note**
>
> 一部の AWS サービスでは、CloudWatch 使用状況メトリクスを使用して、CloudWatch グラフやダッシュボードで現在のサービスの使用状況を可視化できます。CloudWatch のメトリクスの数学関数を使用すると、それらリソースのサービスクォータをグラフ化できます。また、使用量がサービスクォータに近づいたときに警告するアラームも設定できます。詳細については、「サービスクォータの可視化とアラームの設定」を参照してください。

リソース	デフォルトのクォータ
アラームアクション	5/アラーム。このクォータは変更できません。
アラームの評価期間	使用された評価期間数をアラーム期間に乗算して算出される最大値は、1 日 (86,400 秒) です。このクォータは変更できません。
アラーム	10/月/顧客で無料。追加のアラームには料金が発生します。 アカウントあたりのアラームの合計数に制限はありません。 メトリクスの数式に基づくアラームには、最大 10 個のメトリクスを持つことができます。 リージョンごとに 200 個の Metrics Insights アラーム。クォータの引き上げをリクエストできます。
異常検出モデル	アカウントごと、リージョンごとに 500。
API リクエスト	1,000,000/月/顧客で無料。
Canary	リージョンごと、アカウントごとに 200

リソース	デフォルトのクォータ
	クォータの引き上げをリクエストできます。
Contributor Insights API リクエスト	次の API には、リージョンごとに 1 秒あたりのトランザクション (TPS) 数が 20 のクォータがあります。 • DescribeInsightRules クォータは変更できません。 • GetInsightRuleReport クォータの引き上げをリクエストできます。 次の API には、リージョンごとに 5 TPS のクォータがあります。このクォータは変更できません。 • DeleteInsightRules • PutInsightRule 次の API には、リージョンあたり 1 TPS のクォータがあります。このクォータは変更できません。 • DisableInsightRules • EnableInsightRules
Contributor Insights のルール	アカウントあたりリージョンごとに 100 個のルール クォータの引き上げをリクエストできます。
カスタムメトリクス	クォータなし。

リソース	デフォルトのクォータ
ダッシュボード	ダッシュボードあたり最大 500 ウィジェット。ダッシュボードウィジェットあたり最大 500 メトリクス。すべてのウィジェットでダッシュボードあたり最大 2500 メトリクス。 これらのクォータには、メトリクスがグラフに表示されていない場合でも、メトリクス数学関数で使用するために取得されたすべてのメトリクスが含まれます。 これらのクォータは変更できません。
DescribeAlarms	リージョンあたり毎秒 9 トランザクション (TPS) スロットリングなしで 1 秒あたりに実行できるオペレーションリクエストの最大数。 クォータの引き上げをリクエストできます。
DeleteAlarms リクエスト DescribeAlarmHistory リクエスト DisableAlarmActions リクエスト EnableAlarmActions リクエスト SetAlarmState リクエスト	これらのオペレーションごとに、リージョンあたり 3 TPS スロットリングなしで 1 秒あたりに実行できるオペレーションリクエストの最大数。 これらのクォータは変更できません。
DescribeAlarmsForMetric リクエスト	リージョンあたり 9 TPS。スロットリングなしで 1 秒あたりに実行できるオペレーションリクエストの最大数。 このクォータは変更できません。
DeleteDashboards リクエスト GetDashboard リクエスト ListDashboards リクエスト PutDashboard リクエスト	これらのオペレーションごとに、リージョンあたり 10 TPS スロットリングなしで 1 秒あたりに実行できるオペレーションリクエストの最大数。 これらのクォータは変更できません。

リソース	デフォルトのクォータ
PutAnomalyDetector DescribeAnomalyDetectors	リージョンあたり 10 TPS スロットリングなしで 1 秒あたりに実行できるオペレーションリクエストの最大数。
DeleteAnomalyDetector	リージョンあたり 5 TPS スロットリングなしで 1 秒あたりに実行できるオペレーションリクエストの最大数。
ディメンション	30/メトリック。このクォータは変更できません。

リソース	デフォルトのクォータ
GetMetricData	Metrics Insights クエリを含むオペレーションでは、リージョンあたり 10 TPS です。Metrics Insights クエリを含まないオペレーションの場合、クォータはリージョンあたり 50 TPS です。これは、スロットリングなしで 1 秒あたりに実行できるオペレーションリクエストの最大数です。クォータの引き上げをリクエストできます。 Metrics Insights クエリを含む `GetMetricData` オペレーションでは、直近の 3 時間のクォータは 4,300,000 データポイント/秒 (DPS) です。これは、クエリによってスキャンされたデータポイントの総数に対して計算されます (10,000 を超えるメトリクスを含めることはできません)。 API リクエストで使用される `StartTime` が現在の時刻より 3 時間以内である場合は、180,000 Datapoints Per Second (DPS)。`StartTime` が現在の時刻より 3 時間を超える場合は、396,000 DPS。これは、スロットリングなしで 1 つ以上の API コールを使用して 1 秒あたりにリクエストできるデータポイントの上限です。このクォータは変更できません。 DPS は、実際のデータポイントではなく、推定データポイントに基づいて計算されます。推定データポイントは、リクエストされた時間範囲、期間、および保持期間を使用して計算されます。つまり、リクエストされたメトリクスの実際のデータポイントがスパースまたは空の場合でも、推定データポイントがクォータを超えていれば、スロットリングが発生します。DPS クォータはリージョン単位です。

リソース	デフォルトのクォータ
GetMetricData	単一の GetMetricData 呼び出しに以下のものを含めることができます。 • 500 個もの MetricDataQuery 構造体。 • 100 個もの SERVICE_QUOTA() 関数。 • 100 個もの SEARCH() 関数。 • 最大 5 個の LAMBDA() 関数 これらのクォータは変更できません。
GetMetricStatistics	リージョンあたり 400 TPS。スロットリングなしで 1 秒あたりに実行できるオペレーションリクエストの最大数。 クォータの引き上げをリクエストできます。
GetMetricWidgetImage	イメージあたり最大 500 メトリクス。このクォータは変更できません。 リージョンあたり 20 TPS。スロットリングなしで 1 秒あたりに実行できるオペレーションリクエストの最大数。 クォータの引き上げをリクエストできます。
ListMetrics	リージョンあたり 25 TPS スロットリングなしで 1 秒あたりに実行できるオペレーションリクエストの最大数。 クォータの引き上げをリクエストできます。
メトリクスデータ値	メトリクスデータポイントの値は、-2^360 ~ 2^360 の範囲内である必要があります。特殊な値 (NaN、+Infinity、-Infinity など) はサポートされていません。このクォータは変更できません。

リソース	デフォルトのクォータ
MetricDatum 項目	1000/PutMetricData リクエスト。1 つの MetricDatum オブジェクトに単一値、または多数の値を表す StatisticSet オブジェクトを含めることができます。このクォータは変更できません。
メトリクス	10/月/顧客で無料。
Metrics Insights クエリ	1 つのクエリで処理できるメトリクスは 10,000 個以下です。つまり、SELECT 句、FROM 句、および WHERE 句が 10,000 を超えるメトリクスと一致した場合、見つかったこれらのメトリクスのうち最初の 10,000 のみがクエリによって処理されます。 1 つのクエリで 500 を超える時系列を返すことはできません。 直近 3 時間分のデータにのみクエリを実行できます
Observability Access Manager (OAM) API リクエストレート	PutSinkPolicy の場合、リージョンあたり 1 TPS です。 他の各 CloudWatch OAM API の場合、リージョンあたり 10 TPS です。 これらのクォータには、スロットリングなしで 1 秒あたりに実行できるオペレーションリクエストの最大数が反映されます。 これらのクォータは変更できません。
OAM ソースアカウントリンク	各ソースアカウントは、最大 5 つのモニタリングアカウントにリンクできます。 このクォータは変更できません。
OAM シンク	リージョンごと、アカウントごとに 1 つのシンク このクォータは変更できません。

リソース	デフォルトのクォータ
PutCompositeAlarm リクエスト	リージョンあたり 3 TPS スロットリングなしで 1 秒あたりに実行できるオペレーションリクエストの最大数。 クォータの引き上げをリクエストできます。
PutMetricAlarm リクエスト	リージョンあたり 3 TPS スロットリングなしで 1 秒あたりに実行できるオペレーションリクエストの最大数。 クォータの引き上げをリクエストできます。
PutMetricData リクエスト	HTTP POST リクエストでは 1 MB。PutMetricData は、スロットリングなしで 1 秒あたりに実行できるオペレーションリクエストの最大数として、500 件のトランザクション/秒 (TPS) を処理できます。PutMetricData はリクエストごとに 1000 件を処理できます。 クォータの引き上げをリクエストできます。
Amazon SNS メール通知	1,000/月/顧客で無料。
Synthetics グループ	アカウントあたり 20 個。 このクォータは変更できません。
TagResource	リージョンあたり 20 TPS。スロットリングなしで 1 秒あたりに実行できるオペレーションリクエストの最大数。 このクォータは変更できません。
UntagResource	リージョンあたり 20 TPS。スロットリングなしで 1 秒あたりに実行できるオペレーションリクエストの最大数。 このクォータは変更できません。

ドキュメント履歴

次の表に、2018 年 6 月以降の Amazon CloudWatch ユーザーガイドの各リリースにおける重要な変更点を示します。このドキュメントの更新に関する通知を受け取るには、RSS フィードにサブスクライブできます。

変更	説明	日付
CloudWatch Application Signals サービスマップは、Canary、RUM クライアント、AWS サービス依存関係グループ化をサポートしています。	Application Signals のプレビューリリースで、Canary、RUM クライアント、同タイプの AWS サービス依存関係の、サービスマップでのデフォルトのグループ化が追加されました。この変更により、サービスマップのデフォルトビューのアイコン数が減り、閲覧および移動がこれまでよりも容易になりました。	2024 年 5 月 21 日
CloudWatchReadOnlyAccess IAM ポリシーを更新	CloudWatch の、CloudWatchReadOnlyAccess のアクセス許可のスコープが変更されました。ポリシーのスコープに `application-signals:BatchGet*` 、`application-signals:Get*` 、`application-signals:List*` が追加され、ユーザーは、CloudWatch Application Signals を使ってサービスの正常性を表示、調査、診断できるようになりました。CloudWatch に `iam:GetRole` アクションも	2024 年 5 月 17 日

	追加され、Application Signals が設定されているかどうかをユーザーがチェックできるようになりました。	
CloudWatchFullAccessV2 IAM ポリシーを更新	CloudWatch の、CloudWatchFullAccessV2 のアクセス許可のスコープが変更されました。ポリシーのスコープに `application-signals:*` が追加され、ユーザーは、CloudWatch Application Signals を使ってサービスの正常性を表示、調査、診断できるようになりました。	2024 年 5 月 17 日
Lambda Insights が AWS GovCloud (米国東部) と AWS GovCloud (米国西部) をサポート	CloudWatch Lambda Insights では、AWS GovCloud (米国東部) リージョンと AWS GovCloud (米国西部) リージョンのサポートが追加されました。	2024 年 4 月 29 日
CloudWatch クロスアカウントオブザーバビリティがリソースフィルターをサポート	アカウント間のリンクを作成するときに、ソースアカウントからモニタリングアカウントにどのメトリクス名前空間とロググループを共有するかを指定するフィルターを作成できるようになりました。	2024 年 4 月 26 日

CloudWatch Application Signals の更新	Application Signals プレビューリリースに 3 つの機能が追加されました。Application Signals が Python アプリケーションのサポートを開始しました。Amazon EKS アーキテクチャ上のアプリケーションに、より簡単な有効化プロセスを提供します。また、収集されたメトリクスの基数を管理するために使用できる新しい設定も含まれています。	2024 年 4 月 26 日
Amazon EKS 向けにオブザーバビリティが強化された CloudWatch Container Insights が AWS Elastic Fabric Adapter (EFA) メトリクスを収集可能	Amazon EKS 向けにオブザーバビリティが強化された CloudWatch Container Insights を使用して、Amazon EKS クラスターから AWS Elastic Fabric Adapter (EFA) メトリクスを収集できるようになりました。	2024 年 4 月 23 日

IAM ポリシーの更新	CloudWatch では、CloudWatchApplicationSignalsServiceRolePolicy ポリシーが更新されました。このポリシーの `logs:StartQuery` および `logs:GetQueryResults` アクセス許可のスコープが変更されて、より多くのアーキテクチャで Application Signals が有効化されるように `arn:aws:logs:*:*:log-group:/aws/appsignals/*:*` および `"arn:aws:logs:*:*:log-group:/aws/application-signals/data:*"` が追加されました。このポリシーが、AWSServiceRoleForCloudWatchApplicationSignals サービスにリンクされたロールに添付されます。	2024 年 4 月 18 日
Internet Monitor が認証済みの AWS のお客様に世界中のインターネットの状況マップを提供	Amazon CloudWatch Internet Monitor では、認証済みのすべての AWS のお客様がコンソールで利用できる世界中のインターネットの状況マップが表示されるようになりました。マップを表示するには、Amazon CloudWatch コンソールで [インターネットモニター] に移動します。	2024 年 4 月 16 日

Amazon EKS 向けにオブザーバビリティが強化された CloudWatch Container Insights が AWS Neuron メトリクスを収集可能	Amazon EKS 向けにオブザーバビリティが強化された CloudWatch Container Insights を使用して、Amazon EKS クラスターから AWS Neuron メトリクスを収集できるようになりました。	2024 年 4 月 16 日
CloudWatch Application Signals が、診断に役立つように [サービスの概要] タブとその他のメトリクスを追加	新しい [サービスの概要] タブには、オペレーション、依存関係、Synthetics、クライアントページの数など、サービスの概要が表示されます。このタブには、サービス全体の主要なメトリクスと、上位のオペレーションと依存関係が表示されます。また、障害、エラー、遅延などの問題に関連する X-Ray トレースも表示できるようになりました。	2024 年 4 月 16 日
Amazon EKS 向けにオブザーバビリティが強化された CloudWatch Container Insights が Windows のサポートを追加	Amazon EKS 向けにオブザーバビリティが強化された CloudWatch Container Insights を使用して、Amazon EKS クラスターの Windows ワーカーノードからメトリクスを収集できるようになりました。	2024 年 4 月 10 日

CloudWatchApplicationSignalsServiceRolePolicy IAM ポリシーを更新	CloudWatch では、CloudWatchApplicationSignalsServiceRolePolicy のアクセス許可のスコープが変更されました。`cloudwatch:GetMetricData` アクセス許可のスコープが * に変更され、Application Signals が連結アカウントのソースからメトリクスを取得できるようになりました。	2024 年 4 月 8 日
Amazon CloudWatch Internet Monitor で、クロスアカウントオブザーバビリティのサポートを開始	Internet Monitor のクロスアカウントオブザーバビリティを使用して、1 つの AWS リージョン内の複数の AWS アカウントにまたがるアプリケーションをモニタリングできるようになりました。	2024 年 3 月 29 日

CloudWatchAgentServerPolicy ポリシーと CloudWatchAgentAdminPolicy ポリシーを更新	CloudWatch は、CloudWatchAgentServerPolicy ポリシーと CloudWatchAgentAdminPolicy ポリシーの両方にアクセス許可を追加し、CloudWatch エージェントが X-Ray トレースを公開し、ロググループの保持期間を変更できるようにしました。どちらのポリシーでも、xray:PutTraceSegments 、xray:PutTelemetryRecords 、xray:GetSamplingRules 、xray:GetSamplingTargets 、xray:GetSamplingStatisticSummaries および logs:PutRetentionPolicy のアクセス許可が追加されました。	2024 年 2 月 21 日

CloudWatch Network Monitor 向けの新しいサービスリンクロールと IAM ポリシー	CloudWatch に AWSServiceRoleForNetworkMonitor という新しいサービスリンクロールを追加しました。今回新たに追加したサービスリンクロールを使用すると、モニタを作成してソースサブネットと宛先 IP アドレス間のネットワークメトリクスを取得できます。このロールには新しい IAM ポリシー CloudWatchNetworkMonitorServiceRolePolicy がアタッチされています。ユーザーに代わってネットワークメトリクスを取得するアクセス許可を CloudWatch に付与することが目的です。	2023 年 12 月 22 日
CloudWatch で Amazon CloudWatch Network Monitor をリリース	CloudWatch で Amazon CloudWatch Network Monitor という新機能をリリースしました。新しいアクティブなネットワークモニタリングサービスであり、AWS ネットワーク内や社内ネットワークにネットワークの問題があるかどうかを特定できます。	2023 年 12 月 22 日

CloudWatchReadOnlyAccess ポリシーを更新	このポリシーを持つユーザーが CloudWatch Application Signals から報告されたサービス状態の問題をトリアージして診断できるように、CloudWatch Synthetics、X-Ray、CloudWatch RUM に対する既存の読み取り専用アクセス許可と、CloudWatch Application Signals に対する新しい読み取り専用アクセス許可を CloudWatchReadOnlyAccess に追加しました。このポリシーを持つユーザーが自然言語プロンプトから CloudWatch Metrics Insights クエリ文字列を生成できるように、`cloudwatch:GenerateQuery` アクセス許可を追加しました。	2023 年 12 月 5 日
CloudWatchFullAccessV2 ポリシーを更新	このポリシーを持つユーザーが Application Signals を完全に管理して、サービス状態に関する問題をトリアージして診断できるように、CloudWatch Synthetics、X-Ray、CloudWatch RUM に対する既存のアクセス許可を CloudWatchFullAccessV2 に追加したほか、CloudWatch Application Signals に対する新しいアクセス許可を追加しました。	2023 年 12 月 5 日

新しいサービスにリンクされたロールと新しい IAM ポリシー	CloudWatch に AWSServiceRoleForCloudWatchApplicationSignals という新しいサービスリンクロールを追加しました。CloudWatch Application Signals を利用できるアプリケーションから CloudWatch ログデータ、X-Ray トレースデータ、CloudWatch メトリクスデータ、タグデータを収集できるように、この新しいサービスリンクロールを追加しました。このロールには、新しい IAM ポリシー CloudWatchApplicationSignalsServiceRolePolicy がアタッチされています。CloudWatch Application Signals が他の関連 AWS サービスからモニタリングデータとタグデータを収集できるように必要なアクセス許可を付与することが目的です。	2023 年 11 月 30 日

CloudWatch で Application Signals のプレビューリリースを開始	CloudWatch Application Signals はプレビュー中です。Application Signals を使用して AWS でアプリケーションを計測することで、アプリケーションの現在の状態のモニタリング、サービスレベル目標 (SLO) の作成、ビジネス目標に照らした長期にわたるアプリケーションパフォーマンスの追跡が可能になります。詳細については、「Application Signals」を参照してください。	2023 年 11 月 30 日
CloudWatch に他のデータソースをクエリするためのサポートを追加	CloudWatch を使用して、他のデータソースのメトリクスに対してアラームのクエリ、可視化、作成を行うことができます。詳細については、「Querying metrics from other data sources」を参照してください。	2023 年 11 月 26 日
CloudWatch Metrics Insights が自然言語クエリの生成をサポート	CloudWatch Metrics Insights では、自然言語クエリを使用してクエリを生成および更新できます。詳細については、「自然言語を使用した CloudWatch Metrics Insights クエリの生成と更新」を参照してください。	2023 年 11 月 26 日

CloudWatch が Amazon EKS 向けにオブザーバビリティが強化された Container Insights をリリース	CloudWatch では、Container Insights の新しいバージョンがリリースされました。このバージョンでは、Amazon EKS クラスター向けの強化されたオブザーバビリティがサポートされます。また、これらのクラスターからより詳細なメトリクスを収集できます。インストール後は、Amazon EKS クラスターの詳細なインフラストラクチャテレメトリとコンテナログが自動的に収集されます。その後、キュレーションされたすぐに使用できるダッシュボードを使用して、アプリケーションおよびインフラストラクチャテレメトリを掘り下げて調べることができます。	2023 年 11 月 6 日
CloudWatch メトリクスストリームでパートナーの迅速なセットアップが可能に	CloudWatch メトリクスストリームにパートナーの迅速なセットアップオプションが追加されました。これを使用すると、一部のサードパーティプロバイダーへのメトリクスストリームをすばやくセットアップできます。	2023 年 10 月 17 日

CloudWatch がアラームに関する推奨事項をリリース	CloudWatch Synthetics で、他の AWS サービスからのメトリクスに関するアラーム推奨事項が提供されるようになりました。これらの推奨事項は、アラームを設定すべきメトリクスを特定し、これらのサービスのモニタリングにおけるベストプラクティスを遵守するのに役立ちます。	2023 年 10 月 16 日
CloudWatch Synthetics がランタイム syn-nodejs-puppeteer-6.0 をリリース	CloudWatch Synthetics は、ランタイム `syn-nodejs-puppeteer-6.0` をリリースしました。	2023 年 9 月 26 日
クロスアカウントアプリケーションに対する Amazon CloudWatch Application Insights のサポートを追加	アカウントの境界を越えて CloudWatch Application Insights アプリケーションを共有できるようになりました。	2023 年 9 月 26 日

新しいサービスにリンクされたロールと新しい IAM ポリシー	CloudWatch では、AWSServiceRoleForCloudWatchMetrics_DbPerfInsights という新しいサービスにリンクされたロールが追加されました。この新しいサービスにリンクされたロールが追加され、CloudWatch がアラーム、異常検出、スナップショット用の Performance Insights メトリクスを取得できるようにしました。新しい AWSServiceRoleForCloudWatchMetrics_DbPerfInsightsServiceRolePolicy IAM ポリシーがこのロールにアタッチされました。このポリシーはユーザーに代わって Performance Insights メトリクスを取得するアクセス許可を CloudWatch に付与します。	2023 年 9 月 20 日
[新しい Metric Math 関数を追加]	CloudWatch には、アラーム、異常検出、スナップショット作成用に AWS データベースサービスから Performance Insights メトリクスを取得するために使用できる新しい Metric Math 関数 `DB_PERF_INSIGHTS` が追加されました。	2023 年 9 月 20 日

CloudWatchReadOnlyAccess ポリシーを更新	CloudWatch では、このポリシーを持つユーザーが Application Auto Scaling ポリシーに関する情報にアクセスできるように、CloudWatchReadOnlyAccess に `application-autoscaling:DescribeScalingPolicies` アクセス許可が追加されました。	2023 年 9 月 14 日
CloudWatch エージェントで AL2023 のサポートが追加されました	CloudWatch エージェントは AL2023 をサポートします。	2023 年 8 月 8 日
新しいマネージド IAM ポリシー CloudWatchFullAccessv2	CloudWatch に新しいポリシー CloudWatchFullAccessV2 が追加されました。このポリシーは、CloudWatch のアクションとリソースへのフルアクセスを許可すると同時に、Amazon SNS や Amazon EC2 Auto Scaling などの他のサービスに付与されるアクセス許可の範囲をより適切に設定します。	2023 年 8 月 1 日

Amazon CloudWatch Internet Monitor のサービスにリンクされたロールが更新されました — 既存のポリシーへの更新	Internet Monitor のサービスにリンクされたロールに新しいアクセス許可 `elasticloadbalancing:DescribeLoadBalancers` および `ec2:DescribeNetworkInterfaces` を追加し、特定の Network Load Balancer リソースのトラフィック監視をサポートします。	2023 年 7 月 25 日
Amazon CloudWatch Internet Monitor に Network Load Balancer リソースのサポートが追加されました	Internet Monitor に特定の Network Load Balancer リソースを使用してモニターを作成する機能が追加され、アプリケーションのより詳細なオブザーバビリティが可能になります。	2023 年 7 月 25 日

ダッシュボード変数機能	CloudWatch は、ダッシュボード変数をリリースしました。これを使用すると、ダッシュボード内の 1 つの入力フィールドの設定方法に応じて、さまざまなコンテンツをすばやく表示できる柔軟なダッシュボードを作成できます。例えば、さまざまな Lambda 関数または Amazon EC2 インスタンス ID をすばやく切り替えることができるダッシュボードや、さまざまな AWS リージョンに切り替えることができるダッシュボードを作成できます。詳細については、「ダッシュボード変数を使用して柔軟なダッシュボードを作成する」を参照してください。	2023 年 6 月 28 日
Internet Monitor でヘルスイベントのしきい値のカスタマイズのサポートを開始	Internet Monitor は、グローバルパフォーマンススコアまたはアベイラビリティスコアがヘルスイベントをトリガーする場合のしきい値をカスタマイズする機能を追加しました。詳細については、「Amazon CloudWatch Internet Monitor によるパフォーマンスと可用性のリアルタイムでの追跡」を参照してください。	2023 年 6 月 26 日

Internet Monitor ですべての商用リージョンのサポートを開始	Internet Monitor に 7 つの新しい AWS リージョン が追加され、すべての商用リージョンがサポートされるようになりました。	2023 年 6 月 19 日
新しい Lambda Insights の拡張機能のバージョン	CloudWatch は、x86-64 プラットフォームと ARM64 プラットフォームの両方のために、Lambda Insights 拡張機能の 1.0.229.0 バージョンを追加しました。詳細については、「Lambda Insights 拡張機能の利用可能なバージョン」を参照してください。	2023 年 6 月 12 日
CloudWatchReadOnlyAccess ポリシーを更新	CloudWatch が、CloudWatchReadOnlyAccess に対する許可を追加しました。`logs:StartLiveTail` および `logs:StopLiveTail` 許可が追加されたため、このポリシーを持つユーザーがコンソールを使用して CloudWatch Logs ライブテールセッションを開始および停止できるようになりました。詳細については、「ライブテールを使用してほぼリアルタイムでログを表示する」を参照してください。	2023 年 6 月 6 日

CloudWatch RUM がカスタムメトリクスのサポートを追加	CloudWatch RUM アプリケーションモニタを使用してカスタムメトリクスを作成し、CloudWatch と CloudWatch Evidently に送信できます。この機能には、AmazonCloudWatchRUMServiceRolePolicy 管理 IAM ポリシーの更新が含まれます。そのポリシーでは、条件キーが変更されたため、CloudWatch RUM がカスタムメトリクスをカスタムメトリクス名前空間に送信できるようになりました。	2023 年 2 月 9 日
CloudWatch の新しいマネージドポリシーと更新されたマネージドポリシー	CloudWatch クロスアカウントオブザーバビリティをサポートするために、`CloudWatchFullAccess` および `CloudWatchReadOnlyAccess` ポリシーが更新され、新しいマネージドポリシーである `OAMReadOnlyAccess` 、`CloudWatchCrossAccountSharingConfiguration` 、および `OAMFullAccess` が追加されました。詳細については、「CloudWatch の AWS マネージドポリシーへの更新」を参照してください。	2023 年 2 月 7 日

CloudWatch Application Insights サービスリンクロールポリシーの更新 – 既存のポリシーに対する更新です。	CloudWatch Application Insights が既存の AWS サービスリンクロールポリシーを更新しました。	2022 年 12 月 19 日
Amazon CloudWatch Application Insights が Container Insights コンソールからコンテナ化されたアプリケーションとマイクロサービスをサポート。	Container Insights ダッシュボードに、Amazon ECS および Amazon EKS について CloudWatch Application Insights が検出した問題を表示できます。	2021 年 11 月 17 日
Amazon CloudWatch Application Insights による SAP HANA データベースのモニタリング	Application Insights を使用して、SAP HANA データベースをモニタリングできます。	2021 年 11 月 15 日
Amazon CloudWatch Application Insights がアカウント内のすべてのリソースのモニタリングをサポート。	アカウント内のすべてのリソースをオンボードし、モニタリングできます。	2021 年 9 月 15 日
Amazon CloudWatch Application Insights が Amazon FSx をサポート。	Amazon FSx から取得したメトリクスをモニタリングできます。	2021 年 8 月 31 日
SDK メトリクスのサポートは終了しました。	CloudWatch SDK メトリクスのサポートは終了しました。	2021 年 8 月 25 日
コンテナモニタリングをセットアップするための Amazon CloudWatch Application Insights サポート。	Amazon CloudWatch Application Insights では、ベストプラクティスを使用してコンテナをモニタリングできます。	2021 年 5 月 18 日

Metric Streams を一般提供	Metric Streams を使用すると、選択した送信先に対し、継続的に CloudWatch メトリクスをストリーミングできます。詳細については、Amazon CloudWatch ユーザーガイドの「Metric Streams」を参照してください。	2021 年 3 月 31 日
Amazon CloudWatch Application Insights が、Amazon RDS および Amazon EC2 上の Oracle データベースをモニタリングします。	Amazon CloudWatch Application Insights を使用して、Oracle から取得したメトリクスとログをモニタリングできます。	2021 年 1 月 16 日
Lambda Insights を一般提供	CloudWatch Lambda Insights は、 で実行されているサーバーレスアプリケーション用のモニタリングおよびトラブルシューティングソリューションですAWS Lambda 詳細については、Amazon CloudWatch ユーザーガイドの「Lambda Insight の利用」を参照してください。	2020 年 12 月 3 日
Amazon CloudWatch Application Insights が、Prometheus JMX Exporter メトリクスをモニタリングします。	Amazon CloudWatch Application Insights を使用して、Prometheus JMX Exporter から取得したメトリクスをモニタリングできます。	2020 年 11 月 20 日

CloudWatch Synthetics が新しいランタイムバージョンをリリース	CloudWatch Synthetics は、新しいランタイムバージョンをリリースしました。詳細については、Amazon CloudWatch ユーザーガイドの「Canary のランタイムバージョン」を参照してください。	2020 年 9 月 11 日
Amazon CloudWatch Application Insights が、Amazon RDS および Amazon EC2 上の Postgre SQL をモニタリングします。	Amazon RDS または Amazon EC2 で実行されている PostgreSQL で構築されたアプリケーションをモニタリングできます。	2020 年 9 月 11 日
CloudWatch がダッシュボード共有をサポート	組織や AWS アカウント以外のユーザーと CloudWatch ダッシュボードを共有できるようになりました。詳細については、Amazon CloudWatch ユーザーガイドの「CloudWatch ダッシュボードの共有」を参照してください。	2020 年 9 月 10 日
CloudWatch Application Insights のバックエンドで SQL Server を使用して、.NET アプリケーションのモニターをセットアップします	ドキュメントのチュートリアルを使用すると、CloudWatch Application Insights のバックエンドで SQL Server を使用して、.NET アプリケーションのモニターをセットアップできます。	2020 年 8 月 19 日

AWS CloudFormation が、Amazon CloudWatch Application Insights アプリケーションをサポートします。	AWS CloudFormation テンプレートから、主要メトリクスやテレメトリを含む CloudWatch Application Insights モニターリングをアプリケーション、データベース、ウェブサーバーに追加できます。	2020 年 7 月 30 日
Amazon CloudWatch Application Insights が、Aurora for MySQL データベースクラスターをモニタリングします。	Amazon CloudWatch Application Insights を使用して、Aurora for MySQL データベースクラスター (RDS Aurora) をモニタリングできます。	2020 年 7 月 2 日
CloudWatch Contributor Insights の一般提供	CloudWatch Contributor Insights の一般提供が開始されました。ログデータを分析し、コントリビューターデータを表示する時系列を作成できます。トップ N コントリビューター、一意のコントリビューターの合計数、およびそれらの使用状況に関するメトリクスを確認できます。詳細については、Amazon CloudWatch ユーザーガイドの「Contributor Insights を使用した高カーディナリティデータの分析」を参照してください。	2020 年 4 月 2 日

CloudWatch Synthetics のプレビュー公開	CloudWatch Synthetics がプレビュー公開されました。Canary を作成して、エンドポイントと API を監視できます。詳細については、Amazon CloudWatch ユーザーガイドの「Canary の使用」を参照してください。	2019 年 11 月 25 日
CloudWatch Contributor Insights のプレビュー公開	CloudWatch Contributor Insights がプレビュー公開されました。ログデータを分析し、コントリビューターデータを表示する時系列を作成できます。トップ N コントリビューター、一意のコントリビューターの合計数、およびそれらの使用状況に関するメトリクスを確認できます。詳細については、Amazon CloudWatch ユーザーガイドの「Contributor Insights を使用した高カーディナリティデータの分析」を参照してください。	2019 年 11 月 25 日
CloudWatch で ServiceLens 機能を提供開始	ServiceLens は、トレース、メトリクス、ログ、アラームを 1 か所に統合することで、サービスとアプリケーションの監視性を強化します。ServiceLens では、CloudWatch と AWS X-Ray を統合して、アプリケーションのエンドツーエンドのビューが提供されます。	2019 年 11 月 21 日

CloudWatch を使用して AWS のサービスのクォータをプロアクティブに管理する	CloudWatch を使用して AWS のサービスクォータをプロアクティブに管理できます。CloudWatch 使用状況メトリクスにより、アカウントのリソースの使用状況と API オペレーションを視覚的に把握できます。詳細については、Amazon CloudWatch ユーザーガイドから Service Quotas の統合と使用状況メトリクスを参照してください。	2019 年 11 月 19 日
アラームの状態の変化に応じて CloudWatch からイベントを送信する	CloudWatch アラームの状態が変更されると、CloudWatch が Amazon EventBridge にイベントを送信するようになりました。詳細については、Amazon CloudWatch ユーザーガイドの「アラームイベントと EventBridge」を参照してください。	2019 年 10 月 8 日
Container Insights	CloudWatch Container Insights の一般提供が開始されました。コンテナ化されたアプリケーションとマイクロサービスのメトリクスとログを収集、集計、要約できます。詳細については、Amazon CloudWatch ユーザーガイドの「Container Insights の使用」を参照してください。	2019 年 8 月 30 日

Amazon EKS および Kubernetes での Container Insights プレビューメトリクスの更新	Amazon EKS と Kubernetes の Container Insights パブリックプレビューが更新されました。InstanceId がクラスター EC2 インスタンスにディメンションとして含まれるようになりました。これにより、これらのメトリクスで作成されたアラームが EC2 アクション (Stop、Terminate、Reboot、または Recover) をトリガーできるようになります。さらに、ポッドとサービスメトリクスが Kubernetes 名前空間によってレポートされるようになりました。これにより、名前空間ごとのメトリクスのモニタリングとアラームが簡素化されます。	2019 年 8 月 19 日
AWS Systems Manager OpsCenter 統合の更新	CloudWatch Application Insights と Systems Manager OpsCenter を統合する方法に関する更新。	2019 年 8 月 7 日
CloudWatch 使用状況メトリクス	CloudWatch 使用状況メトリクスは、CloudWatch リソースの使用状況を追跡し、サービスの制限内に抑えるのに役立ちます。詳細については、「https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/CloudWatch-Usage-Metrics.html」を参照してください。	2019 年 8 月 6 日

CloudWatch Container Insights のプレビュー公開	CloudWatch Container Insights がプレビュー公開されました。コンテナ化されたアプリケーションとマイクロサービスのメトリクスとログを収集、集計、要約できます。詳細については、Amazon CloudWatch ユーザーガイドの「Container Insights の使用」を参照してください。	2019 年 7 月 9 日
CloudWatch 異常検出パブリックプレビュー	CloudWatch 異常検出がプレビュー公開されました。CloudWatch は、メトリクスの過去のデータに機械学習アルゴリズムを適用して、メトリクスの期待値のモデルを作成します。このモデルは、可視化やアラームの設定に使用できます。詳細については、Amazon CloudWatch ユーザーガイドの「CloudWatch 異常検出の使用」を参照してください。	2019 年 7 月 9 日
CloudWatch Application Insights for .NET and SQL Server	CloudWatch Application Insights for .NET and SQL Server を使用すると、.NET と SQL Server アプリケーションの監視が容易になります。アプリケーションのリソースを監視する最適な条件を設定し、データを継続的に分析してアプリケーションの問題の徴候を検出できます。	2019 年 6 月 21 日

CloudWatch エージェントセクションの再編成	CloudWatch エージェントのドキュメントは、特にコマンドラインを使用してエージェントをインストールおよび設定しているお客様に対する明確さを高めるように書き直されました。詳細については、Amazon CloudWatch ユーザーガイドの「CloudWatch エージェントを使用して Amazon EC2 インスタンスとオンプレミスサーバーからメトリクスとログを収集する」を参照してください。	2019 年 3 月 28 日
メトリクス数式に SEARCH 関数を追加	メトリクス数式で SEARCH 関数を使用できるようになりました。これにより、検索クエリに一致する新しいリソースが作成されると自動的に更新されるダッシュボードを作成することができます。詳細については、Amazon CloudWatch ユーザーガイドの「グラフで検索式を使用する」を参照してください。	2019 年 3 月 21 日

エンタープライズサポートの AWS SDK メトリクス	SDK メトリクスは、AWS サービスのヘルスを評価し、アカウントの使用制限への到達やサービス停止などによるレイテンシーを診断する場合に役立ちます。詳細については、Amazon CloudWatch ユーザーガイドの AWS SDK メトリクスを使用したアプリケーションのモニタリングを参照してください。	2018 年 12 月 11 日
数式に基づくアラーム	CloudWatch では、メトリクスの数式に基づくアラームを作成できます。詳細については、Amazon CloudWatch ユーザーガイドの「数式に基づくアラーム」を参照してください。	2018 年 11 月 20 日
新しい CloudWatch コンソールのホームページ	Amazon は CloudWatch コンソールの新しいホームページを作成しました。このホームページには、使用している AWS の各サービスの主要なメトリクスとアラームが自動的に表示されます。詳細については、Amazon CloudWatch ユーザーガイドの「Amazon CloudWatch の開始方法」を参照してください。	2018 年 11 月 19 日

CloudWatch エージェントの AWS CloudFormation テンプレート	Amazon は、CloudWatch エージェントのインストールと更新に役立つ複数の AWS CloudFormation テンプレートをアップロードしています。詳細については、Amazon CloudWatch ユーザーガイドの AWS CloudFormation を使用して新しいインスタンスに CloudWatch エージェントをインストールするを参照してください。	2018 年 11 月 9 日
CloudWatch エージェントの機能強化	CloudWatch エージェントは、StatsD プロトコルと collectd プロトコルの両方で動作するように更新されました。また、クロスアカウントのサポートも強化されました。詳細については、Amazon CloudWatch ユーザーガイドの StatsD を使用したカスタムメトリクスの取得、collectd を使用したカスタムメトリクスの取得、別の AWS アカウントへのメトリクスおよびログの送信を参照してください。	2018 年 9 月 28 日
Amazon VPC エンドポイントのサポート	これで、VPC と CloudWatch との間でプライベート接続を確立できます。詳細については、Amazon CloudWatch ユーザーガイドの「インターフェイス VPC エンドポイントでの CloudWatch の使用」を参照してください。	2018 年 6 月 28 日

次の表に、2018 年 6 月以前の Amazon CloudWatch ユーザーガイドの重要な変更点を示します。

変更	説明	リリース日
Metric Math	CloudWatch メトリクスで数式を実行し、ダッシュボードでグラフに追加できる新しい時系列を生成できるようになりました。詳細については、「Metric Math を使用する」を参照してください。	2018 年 4 月 4 日
「N 個中 M 個」のアラーム	アラーム評価間隔中、「N 個中 M 個」のデータポイントに基づいてトリガーするようにアラームを設定できるようになりました。詳細については、「アラームの評価」を参照してください。	2017 年 12 月 8 日
CloudWatch エージェント	新しい統合 CloudWatch エージェントがリリースされました。統一された複数プラットフォームエージェントを使用して、Amazon EC2 インスタンスとオンプレミスサーバーからカスタムのシステムメトリクスとログファイルの両方を収集できます。新しいエージェントでは Windows と Linux の両方がサポートされ、CPU あたりのコアのようなサブリソースメトリクスなど、収集するメトリクスをカスタマイズできます。詳細については、「CloudWatch エージェントを使用してメトリクス、ログ、トレースを収集する」を参照してください。	2017 年 9 月 7 日
NAT ゲートウェイメトリクス	Amazon VPC NAT ゲートウェイのメトリクスを追加しました。	2017 年 9 月 7 日
高解像度のメトリクス	オプションでカスタムメトリクスを高解像度メトリクスとして設定し、詳細度は最小 1 秒を指定できます。詳細については、「高解像度のメトリクス」を参照してください。	2017 年 7 月 26 日
ダッシュボード API	API と を使用して、ダッシュボードを作成、変更、および削除できるようになりましたAWS CLI	2017 年 7 月 6 日

変更	説明	リリース日
	詳細については、「CloudWatch ダッシュボードの作成」を参照してください。	
AWS Direct Connect のメトリクス	追加された のメトリクスAWS Direct Connect	2017 年 6 月 29 日
Amazon VPC VPN メトリクス	Amazon VPC VPN メトリクスを追加しました。	2017 年 5 月 15 日
AppStream 2.0 メトリクス	AppStream 2.0 のメトリクスを追加しました。	2017 年 3 月 8 日
CloudWatch コンソールカラーピッカー	ダッシュボードウィジェットで各メトリックの色を今すぐ選択できます。詳細については、「CloudWatch ダッシュボードでグラフを編集する」を参照してください。	2017 年 2 月 27 日
ダッシュボードのアラーム	アラームはダッシュボードに今すぐ追加できます。詳細については、「CloudWatch ダッシュボードでアラームウィジェットを追加または削除する」を参照してください。	2017 年 2 月 15 日
Amazon Polly のメトリクスを追加	Amazon Polly のメトリクスを追加しました。	2016 年 12 月 1 日
Amazon Managed Service for Apache Flink のメトリクスを追加	Amazon Managed Service for Apache Flink のメトリクスを追加しました。	2016 年 12 月 1 日
パーセンタイル統計のサポートの追加	小数点以下最大 2 桁を使用して、任意のパーセンタイルを指定できます (p95.45 など)。詳細については、「パーセンタイル」を参照してください。	2016 年 11 月 17 日

変更	説明	リリース日
Amazon Simple Email Service のメトリクスを追加	Amazon Simple Email Service のメトリクスを追加しました。	2016 年 11 月 2 日
更新したメトリクスの保持	Amazon CloudWatch では、14 日間ではなく、15 か月の間メトリクスデータが保持されるようになりました。	2016 年 11 月 1 日
更新されたメトリクスコンソールインターフェイス	CloudWatch コンソールは既存の機能と新しい機能をよりよくすることで、更新されます。	2016 年 11 月 1 日
Amazon Elastic Transcoder のメトリクスを追加	Amazon Elastic Transcoder のメトリクスを追加しました。	2016 年 9 月 20 日
Amazon API Gateway のメトリクスを追加	Amazon API Gateway のメトリクスを追加しました。	2016 年 9 月 9 日
追加された のメトリクスAWS Key Management Service	追加された のメトリクスAWS Key Management Service	2016 年 9 月 9 日
Elastic Load Balancing でサポートされる新しい Application Load Balancers のメトリクスを追加	Application Load Balancers のメトリクスを追加しました。	2016 年 8 月 11 日

変更	説明	リリース日
Amazon EC2 用の新しい NetworkPacketsIn と NetworkPacketsOut のメトリクスを追加しました。	Amazon EC2 用の新しい NetworkPacketsIn と NetworkPacketsOut のメトリクスを追加しました。	2016 年 3 月 23 日
Amazon EC2 スポットフリートの新しいメトリクスを追加	Amazon EC2 スポットフリートの新しいメトリクスを追加しました。	2016 年 3 月 21 日
新しい CloudWatch Logs メトリクスを追加	新しい CloudWatch Logs メトリクスを追加しました。	2016 年 3 月 10 日
Amazon OpenSearch Service および AWS WAF メトリクスとディメンションを追加しました	Amazon OpenSearch Service および AWS WAF メトリクスとディメンションを追加しました。	2015 年 10 月 14 日
CloudWatch ダッシュボードのサポートを追加	ダッシュボードは、異なるリージョンに広がっていても、1 つの画面でリソースをモニタリングできる CloudWatch コンソールにあるカスタマイズ可能なホームページです。詳細については、「Amazon CloudWatch ダッシュボードの使用」を参照してください。	2015 年 10 月 8 日

変更	説明	リリース日
AWS Lambda のメトリクスとディメンションを追加	AWS Lambda のメトリクスとディメンションが追加されました。	2015 年 9 月 4 日
Amazon Elastic Container Service のメトリクスとディメンションを追加	Amazon Elastic Container Service のメトリクスとディメンションを追加しました。	2015 年 8 月 17 日
Amazon Simple Storage Service のメトリクスとディメンションを追加	Amazon Simple Storage Service のメトリクスとディメンションを追加しました。	2015 年 7 月 26 日
新機能: アラームアクションの再起動	再起動アラームアクションと、アラームアクションで使用する新しい IAM ロールが追加されました。詳細については、「EC2 インスタンスを停止、終了、再起動、または復旧するアラームを作成する」を参照してください。	2015 年 7 月 23 日
Amazon WorkSpaces のメトリックスとディメンションが追加されました。	Amazon WorkSpaces のメトリックスとディメンションが追加されました。	2015 年 4 月 30 日
Amazon Machine Learning のメトリクスとディメンションを追加	Amazon Machine Learning のメトリクスとディメンションを追加しました。	2015 年 4 月 9 日

変更	説明	リリース日
新機能: Amazon EC2 インスタンスの復旧アラームアクション	アラームアクションが更新され、新しい EC2 インスタンス復旧アクションが含まれるようになりました。詳細については、「EC2 インスタンスを停止、終了、再起動、または復旧するアラームを作成する」を参照してください。	2015 年 3 月 12 日
Amazon CloudFront と Amazon CloudSearch のメトリクスとディメンションを追加	Amazon CloudFront と Amazon CloudSearch のメトリクスとディメンションを追加しました。	2015 年 3 月 6 日
Amazon Simple Workflow Service のメトリクスとディメンションを追加	Amazon Simple Workflow Service のメトリクスとディメンションを追加しました。	2014 年 5 月 9 日
のサポート追加に関するガイドを更新しました AWS CloudTrail	Amazon CloudWatch のアクティビティをログに記録するために AWS CloudTrail を使用する方法について説明する新しいトピックを追加しました。詳細については、「AWS CloudTrail を使用した Amazon CloudWatch API コールのログ記録」を参照してください。	2014 年 4 月 30 日

変更	説明	リリース日
新しい AWS Command Line Interface (AWS CLI) の使用ガイド	AWS CLI は複数のサービス間で用いることができる CLI であり、簡潔なインストール、統合された設定、整合性のあるコマンドライン構文が特長です。AWS CLI は Linux/Unix、Windows、および Mac でサポートされます。このガイドに示す CLI の例は、新しい AWS CLI を用いることができるように更新されました。 新しい AWS CLI をインストールして設定する方法については、AWS Command Line Interface ユーザーガイドの AWS CLI インターフェイスの設定を参照してください。	2014 年 2 月 21 日
Amazon Redshift と AWS OpsWorks のメトリクスとディメンションを追加しました	Amazon Redshift と AWS OpsWorks のメトリクスとディメンションを追加しました。	2013 年 7 月 16 日
Amazon Route 53 のメトリクスとディメンションを追加しました	Amazon Route 53 のメトリクスとディメンションを追加しました。	2013 年 6 月 26 日
新機能: Amazon CloudWatch アラームアクション	ドキュメント「Amazon CloudWatch アラームアクション」に新しいセクションが追加されました。Amazon Elastic Compute Cloud インスタンスの停止または終了に参照できます。詳細については、「EC2 インスタンスを停止、終了、再起動、または復旧するアラームを作成する」を参照してください。	2013 年 1 月 8 日

変更	説明	リリース日
EBS メトリクスを更新	プロビジョニングされた IOPS ボリューム用の 2 つの新しいメトリクスを含むように EBS メトリクスを更新しました。	2012 年 11 月 20 日
新しい請求アラート	Amazon CloudWatch メトリクスを用いて AWS の変化をモニタリングするとともに、アラームを作成して指定されたしきい値を超えた場合に通知できるようになりました。詳細については、「AWS の予想請求額をモニタリングする請求アラームの作成」を参照してください。	2012 年 5 月 10 日
新しいメトリクス	さまざまな HTTP 応答コードをカウントできる 6 つの新しい Elastic Load Balancing メトリクスが利用できるようになりました。	2011 年 10 月 19 日
新機能	Amazon EMR からメトリクスにアクセスできるようになりました。	2011 年 6 月 30 日
新機能	Amazon Simple Notification Service および Amazon Simple Queue Service からメトリクスにアクセスできるようになりました。	2011 年 7 月 14 日
新機能	`PutMetricData` API を使用したカスタムメトリクスのパブリッシュに関する情報を追加しました。詳細については、「カスタムメトリクスをパブリッシュする」を参照してください。	2011 年 5 月 10 日
更新したメトリクスの保持	Amazon CloudWatch によるアラーム履歴の保持期間が 6 週間から 2 週間になりました。この変更により、アラームの保持期間がメトリクスデータの保持期間と一致するようになりました。	2011 年 4 月 7 日
新機能	メトリクスがしきい値を超えたときに Amazon Simple Notification Service または Auto Scaling 通知を送信する機能が追加されました。詳細については、「アラーム」を参照してください。	2010 年 12 月 2 日

変更	説明	リリース日
新機能	多数の CloudWatch アクションに MaxRecords パラメーターと NextToken パラメーターが含められ、表示する結果ページを制御できるようになりました。	2010 年 12 月 2 日
新機能	このサービスが AWS Identity and Access Management (IAM) と連動するようになりました。	2010 年 12 月 2 日