デベロッパーガイド

AWS IoT Events

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

AWS IoT Events: デベロッパーガイド

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

Amazon の商標およびトレードドレスは、Amazon 以外の製品またはサービスに関連して、顧客間で混乱を引き起こす可能性のある方法で、または Amazon を軽蔑または信用を傷つける方法で使用することはできません。Amazon が所有していないその他のすべての商標は Amazon との提携、関連、支援関係の有無にかかわらず、それら該当する所有者の資産です。

Table of Contents

... viii
とは AWS IoT Events ... 1
利点と特徴 ... 1
ユースケース ... 2
リモートデバイスのモニタリングと保守 ... 3
産業用ロボットを管理する ... 3
構築自動化システムを追跡する ... 3
AWS IoT Events サポート終了 ... 4
から移行する際の考慮事項 AWS IoT Events ... 4
デテクターモデル ... 5
アーキテクチャの比較 ... 5
ステップ 1: (オプション) AWS IoT Events ディテクターモデル設定をエクスポートする ... 6
ステップ 2: IAM ロールを作成する ... 7
ステップ 3: Amazon Kinesis Data Streams を作成する ... 10
ステップ 4: MQTT メッセージルーティングルールを作成または更新する ... 10
ステップ 5: 送信先 MQTT トピックのエンドポイントを取得する ... 12
ステップ 6: Amazon DynamoDB テーブルを作成する ... 13
ステップ 7: AWS Lambda 関数を作成する (コンソール) ... 13
ステップ 8: Amazon Kinesis Data Streams トリガーを追加する ... 21
ステップ 9: データの取り込みと出力機能をテストする (AWS CLI) ... 22
アラーム ... 23
アーキテクチャの比較 ... 23
ステップ 1: アセットプロパティで MQTT 通知を有効にする ... 24
ステップ 2: AWS Lambda 関数を作成する ... 25
ステップ 3: AWS IoT Core メッセージルーティングルールを作成する ... 26
ステップ 4: CloudWatch メトリクスを表示する ... 27
ステップ 5: CloudWatch アラームを作成する ... 28
ステップ 6: (オプション) CloudWatch アラームを にインポートする AWS IoT SiteWise ... 28
設定 ... 29
のセットアップ AWS アカウント ... 29
にサインアップする AWS アカウント ... 29
管理アクセスを持つユーザーを作成する ... 30
のアクセス許可の設定 AWS IoT Events ... 31
アクション許可 ... 32

入力データの保護 34
Amazon CloudWatch ログ記録ロール ポリシー 34
Amazon SNS メッセージングロールポリシー 36
入門 38
前提条件 40
入力の作成 41
JSON 入力ファイルを作成する 41
入力の作成と設定 41
ディテクターモデル内に入力を作成する 42
ディテクターモデルの作成 43
ディテクターモデルのテスト 50
ベストプラクティス 54
AWS IoT Events ディテクターモデルの開発時に Amazon CloudWatch ログ記録を有効にする 54
AWS IoT Events コンソールで作業するときにディテクターモデルを保存するために定期的に公開する 55
チュートリアル 56
AWS IoT Events を使用して IoT デバイスを監視する 56
ディテクターモデルに必要な状態をどのようにして知ることができますか？ 58
ディテクターのインスタンスが 1 つ必要か、それとも複数必要かをどのようにして知ることができますか？ 59
簡単なステップバイステップの例 60
デバイスデータをキャプチャするための入力を作成します 62
デバイスの状態を表すディテクターモデルを作成する 63
ディテクターへの入力としてメッセージを送信する 67
ディテクターモデルの制限と制限 70
コメント例: HVAC 温度制御 74
ディテクターモデルの入力定義 75
ディテクターモデル定義を作成する 78
BatchUpdateDetector を使用する 98
入力に BatchPutMessage を使用する 100
MQTT メッセージを取り込む 103
Amazon SNS メッセージを生成する 104
DescribeDetector API を設定する 105
AWS IoT Core ルールエンジンを使用する 107
サポートされているアクション 111

組み込みアクションを使用する 112
タイマーアクションの設定 112
タイマーアクションのリセット 112
タイマーアクションをクリア 113
可変アクションの設定 113
他の AWS サービスの使用 114
AWS IoT Core 115
AWS IoT Events 116
AWS IoT SiteWise 117
Amazon DynamoDB 120
Amazon DynamoDB (v2) 122
Amazon Data Firehose 123
AWS Lambda 124
Amazon Simple Notification Service 125
Amazon Simple Queue Service 126
表現 129
デバイスデータをフィルタリングする構文 129
リテラル 129
演算子 129
式の関数 131
式の入力と変数のリファレンス 135
置換テンプレート 138
使用方法 139
AWS IoT Events 式の記述 139
ディテクターモデルの例 141
HVAC (空気調和工学) の温度制御 141
背景 141
入力定義 142
ディテクターモデルの定義 144
BatchPutMessage の例 162
BatchUpdateDetector の例 168
AWS IoT Core ルールエンジン 170
クレーン 173
コマンドの送信 174
デテクターモデル 175
入力 182

メッセージ ... 182
例: センサーによるイベント検出 ... 184
デバイス HeartBeat ... 186
ISA アラーム ... 189
シンプルなアラーム ... 199
アラームを使用したモニタリング ... 204
の使用 AWS IoT SiteWise ... 204
フローの確認 ... 204
アラームモデルの作成 ... 205
要件 ... 206
アラームモデルの作成 (コンソール) ... 206
アラームへの対応。 ... 209
アラーム通知の管理 ... 210
Lambda 関数の作成 ... 211
Lambda 関数の使用 ... 220
アラーム受信者を管理する ... 221
セキュリティ ... 223
Identity and Access Management ... 223
対象者 ... 224
アイデンティティを使用した認証 ... 225
ポリシーを使用したアクセスの管理 ... 228
ID とアクセス管理の詳細 ... 230
と IAM の AWS IoT Events 連携方法 ... 230
アイデンティティベースのポリシーの例 ... 234
のサービス間の混乱した代理の防止 AWS IoT Events ... 240
トラブルシューティング ... 244
モニタリング ... 246
モニタリングできるツール AWS IoT Events ... 247
Amazon CloudWatch AWS IoT Events によるモニタリング ... 248
を使用した AWS IoT Events API コールのログ記録 AWS CloudTrail ... 250
コンプライアンス検証 ... 269
耐障害性 ... 270
インフラストラクチャセキュリティ ... 271
クォータ ... 272
Tagging ... 273
ベーシックタグ ... 273

タグの制約と制限 274
IAM ポリシーでのタグの使用 274
トラブルシューティング 278
一般的な AWS IoT Events 問題と解決策 278
ディテクターモデルの作成エラー 279
削除されたディテクターモデルからの更新 279
アクショントリガーの障害 (条件を満たす場合) 279
アクショントリガーの障害 (しきい値を超えた場合) 280
誤った状態の使用法 280
接続メッセージ 280
InvalidRequestException メッセージ 281
Amazon CloudWatch Logs `action.setTimer` エラー 281
Amazon CloudWatch ペイロードエラー 282
互換性のないデータ型 283
へのメッセージの送信に失敗しました AWS IoT Events 285
ディテクターモデルのトラブルシューティング 285
診断情報 286
ディテクターモデルを分析する (コンソール) 299
ディテクターモデルを分析する (AWS CLI) 301
コマンド 306
AWS IoT Events アクション 306
AWS IoT Events データ 306
ドキュメント履歴 307
以前の更新 308

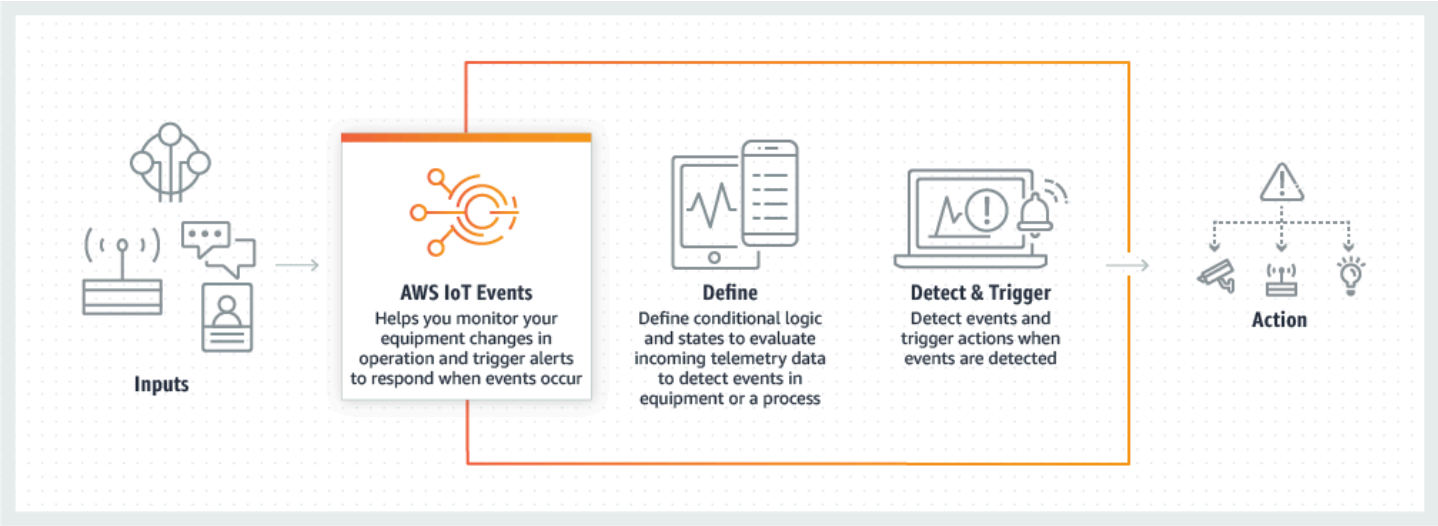
サポート終了通知: 2026 年 5 月 20 日に、 AWS は のサポートを終了します AWS IoT Events。2026 年 5 月 20 日以降、 AWS IoT Events コンソールまたは AWS IoT Events リソースにアクセスできなくなります。詳細については、AWS IoT Events 「サポート終了」を参照してください。

翻訳は機械翻訳により提供されています。提供された翻訳内容と英語版の間で齟齬、不一致または矛盾がある場合、英語版が優先します。

とは AWS IoT Events

AWS IoT Events を使用すると、機器またはデバイスフリートのオペレーションの障害や変更をモニタリングし、そのようなイベントが発生したときにアクションをトリガーできます。 は、デバイス、プロセス、アプリケーション、およびその他の AWS サービスの IoT センサーデータ AWS IoT Events を継続的に監視して、重要なイベントを識別し、アクションを実行できるようにします。

を使用して AWS IoT Events 、 AWS IoT Events コンソールまたは API からアクセスできる複雑なイベントモニタリングアプリケーションを AWS クラウドに構築します。 APIs



トピック

- 利点と特徴
- ユースケース

利点と特徴

複数のソースからの入力を受け入れる

AWS IoT Events は、多くの IoT テレメトリデータソースからの入力を受け入れます。これには、センサーデバイス、管理アプリケーション、 AWS IoT Core や などの他の AWS IoT サービスが含まれます AWS IoT Analytics。標準 API インターフェイス (`BatchPutMessage` API) または AWS IoT Events コンソール AWS IoT Events を使用して、テレメトリデータ入力を にプッシュできます。

の使用開始の詳細については AWS IoT Events、「」を参照してください[AWS IoT Events コンソールの開始方法](#)。

単純な論理式を使用してイベントの複雑なパターンを認識する

AWS IoT Events は、単一の IoT デバイスまたはアプリケーションからの複数の入力、または多様な機器や多数の独立したセンサーからの複数の入力を含むイベントのパターンを認識できます。これは、各センサーとアプリケーションが重要な情報を提供するため、特に便利です。しかし、多様なセンサーおよびアプリケーションデータを組み合わせることによってのみ、オペレーションのパフォーマンスおよび品質を完全に把握できます。複雑なコードの代わりに単純な論理式を使用してこれらのイベントを認識するように AWS IoT Events ディテクターを設定できます。

論理式の詳細については、「」を参照してください[イベントデータをフィルタリング、変換、処理する式](#)。

イベントに基づいてアクションをトリガーする

AWS IoT Events を使用すると、Amazon Simple Notification Service (Amazon SNS)、Lambda AWS IoT Core、Amazon SQS、Amazon Kinesis Firehose でアクションを直接トリガーできます。また、Amazon Connect などの他の サービスや独自のエンタープライズリソースプランニング (ERP) アプリケーションを使用してアクションを実行できるようにする AWS IoT ルールエンジンを使用して AWS Lambda 関数をトリガーすることもできます。

AWS IoT Events には、実行できるアクションのビルド済みライブラリが含まれており、独自のアクションを定義することもできます。

イベントに基づくアクションのトリガーの詳細については、「」を参照してください[でデータを受信してアクションをトリガーするためにサポートされているアクション AWS IoT Events](#)。

フリートの需要に合わせて自動的にスケーリングする

AWS IoT Events は、同種デバイスを接続するときに自動的にスケーリングします。特定のタイプのデバイスに対してディテクターを 1 回定義すると、サービスは自動的にその接続先のデバイスのすべてのインスタンスをスケーリングおよび管理します AWS IoT Events。

ディテクターモデルの例については、「」を参照してください[AWS IoT Events ディテクターモデルの例](#)。

ユースケース

AWS IoT Events には多くの用途があります。ユースケースの例をいくつか示します。

リモートデバイスのモニタリングと保守

リモートでデプロイされたマシンのフリートのモニタリングは、特に明確なコンテキストなしで誤動作が発生した場合は、困難になる可能性があります。1 台のマシンが機能しなくなった場合は、処理ユニットまたはマシン全体を置き換えることを意味します。しかし、これは持続可能ではありません。 AWS IoT Events を使用すると、各マシンの複数のセンサーからメッセージを受信して、時間の経過とともに特定の問題を診断できます。ユニット全体を置き換える代わりに、置き換えが必要な正確な部分を技術者に送信するために必要な情報が得られるようになりました。数百万台のマシンを使用すると、コスト削減は合計数百万ドルになり、各マシンの所有または保守にかかる総コストが削減されます。

産業用ロボットを管理する

施設にロボットをデプロイしてパッケージの移動を自動化すると、効率が大幅に向上します。コストを最小限に抑えるために、ロボットにはデータをクラウドに報告するシンプルで低コストのセンサーが搭載されています。ただし、数十のセンサーと数百の運用モードでは、リアルタイムで問題を検出するのは難しい場合があります。を使用すると AWS IoT Events、クラウドでこのセンサーデータを処理するエキスパートシステムを構築し、障害が差し迫った場合に技術スタッフに自動的に通知するためのアラートを作成できます。

構築自動化システムを追跡する

データセンターでは、高温度と低湿度をモニタリングすることで、機器の障害を防ぐことができます。センサーは多くの場合、多くの製造元から購入され、各タイプには独自の管理ソフトウェアが付属しています。ただし、異なるベンダーの管理ソフトウェアには互換性がないため、問題の検出が困難になることがあります。を使用すると AWS IoT Events、障害発生前に、運用アナリストに加熱および冷却システムに関する問題を通知するアラートを設定できます。このようにして、機器の交換に数千ドルのコストがかかり、収益が失われる可能性のあるデータセンターの予定外のシャットダウンを防ぐことができます。

AWS IoT Events サポート終了

慎重に検討した結果、2026 年 5 月 20 日以降、 AWS IoT Events サービスのサポートを終了することにしました。 AWS IoT Events は 2025 年 5 月 20 日以降、新規顧客を受け入れなくなります。2025 年 5 月 20 日より前にサービスにサインアップしたアカウントを持つ既存のお客様は、引き続き AWS IoT Events 機能を使用できます。2026 年 5 月 20 日以降、 を使用することはできません AWS IoT Events。

このページでは、ビジネスニーズに合わせて代替ソリューションに移行 AWS IoT Events するための手順と考慮事項について説明します。

> **Note**
>
> これらのガイドで紹介するソリューションは、実稼働準備が整った AWS IoT Events 機能の代替品としてではなく、実例となることを目的としています。ビジネスニーズに合わせてコード、ワークフロー、関連 AWS リソースをカスタマイズします。

トピック

- から移行する際の考慮事項 AWS IoT Events
- のディテクターモデルの移行手順 AWS IoT Events
- での AWS IoT SiteWise アラームの移行手順 AWS IoT Events

から移行する際の考慮事項 AWS IoT Events

- 各コンポーネントの最小特権を持つ IAM ロールの使用や、保管中および転送中のデータの暗号化など、セキュリティのベストプラクティスを実装します。詳細については、「IAM ユーザーガイド」の「IAM でのセキュリティベストプラクティス」を参照してください。
- データ取り込み要件に基づいて、Kinesis ストリームのシャードの数を検討してください。Kinesis シャードの詳細については、Amazon Kinesis Data Streams デベロッパーガイド」の「Amazon Kinesis Data Streams の用語と概念」を参照してください。 Amazon Kinesis
- CloudWatch を使用してメトリクスとログの包括的なモニタリングとデバッグを設定します。詳細については、「Amazon CloudWatch ユーザーガイド」の「CloudWatch とは」を参照してください。 Amazon CloudWatch

- 繰り返し処理に失敗するメッセージを管理する方法、再試行ポリシーの実装、問題のあるメッセージを分離して分析するプロセスの設定など、エラー処理の構造を検討してください。
- AWS 料金計算ツールを使用して、特定のユースケースのコストを見積もります。

のディテクターモデルの移行手順 AWS IoT Events

このセクションでは、移行時に同様のディテクターモデル機能を提供する代替ソリューションについて説明します AWS IoT Events。

AWS IoT Core ルールによるデータインジェストは、他の AWS サービスの組み合わせに移行できます。BatchPutMessage API によるデータ取り込みの代わりに、データを AWS IoT Core MQTT トピックにルーティングできます。

この移行アプローチでは、 AWS IoT Core MQTT トピックを IoT データのエントリポイントとして活用し、 への直接入力を置き換えます AWS IoT Events。MQTT トピックは、いくつかの主な理由で選択されます。MQTT は業界で広く使用されているため、IoT デバイスとの幅広い互換性を提供します。これらのトピックでは、多数のデバイスからの大量のメッセージを処理できるため、スケーラビリティが確保されます。また、コンテンツまたはデバイスタイプに基づいてメッセージのルーティングとフィルタリングを柔軟に行うことができます。さらに、 AWS IoT Core MQTT トピックは他の AWS サービスとシームレスに統合されるため、移行プロセスが容易になります。

MQTT トピックから Amazon Kinesis Data Streams、 AWS Lambda 関数、Amazon DynamoDB テーブル、Amazon EventBridge スケジュールを組み合わせたアーキテクチャにデータが流れます。このサービスの組み合わせにより、 によって以前に提供された機能がレプリケートおよび強化され AWS IoT Events、IoT データ処理パイプラインをより柔軟に制御できます。

アーキテクチャの比較

現在の AWS IoT Events アーキテクチャは、 AWS IoT Core ルールと `BatchPutMessage` API を通じてデータを取り込みます。このアーキテクチャでは、 AWS IoT Core を使用してデータの取り込みとイベント発行を行います。メッセージは、状態ロジックを定義するディテクターモデルへの AWS IoT Events 入力を介してルーティングされます。IAM ロールは、必要なアクセス許可を管理します。

新しいソリューションは、データインジェスト (現在は専用の入出力 MQTT トピックを使用) AWS IoT Core のために を維持します。データパーティショニング用の Kinesis Data Streams と、状態ロジック用の評価者 Lambda 関数が導入されました。デバイスの状態が DynamoDB テーブルに保存され、拡張 IAM ロールがこれらのサービス全体のアクセス許可を管理するようになりました。

目的	ソリューション	相違点
データ取り込み — IoT デバイスからデータを受信します	AWS IoT Core	2 つの異なる MQTT トピックが必要になりました。1 つはデバイスデータの取り込み用、もう 1 つは出力イベントの発行用です。
メッセージの方向 – 受信メッセージを適切なサービスにルーティングします	AWS IoT Core メッセージルーティングルール	同じルーティング機能を維持しますが、 ではなく Kinesis Data Streams にメッセージを送信するようになりました。 AWS IoT Events
データ処理 – 受信データストリームを処理および整理します	Kinesis Data Streams	AWS IoT Events 入力機能を置き換え、データインジェストをメッセージ処理用のデバイス ID パーティショニングに置き換えます
ロジック評価 – 状態の変更を処理し、アクションをトリガーします	評価者 Lambda	AWS IoT Events ディテクターモデルを置き換え、ビジュアルワークフローの代わりにコードを通じてカスタマイズ可能な状態ロジック評価を提供
状態管理 – デバイスの状態を維持します	DynamoDB テーブル	デバイス状態の永続的ストレージを提供し、内部 AWS IoT Events 状態管理を置き換える新しいコンポーネント
セキュリティ – サービスアクセス許可を管理します	IAM ロール	更新されたアクセス許可には、既存のアクセス AWS IoT Core 許可に加えて、Kinesis Data Streams、DynamoDB、EventBridge へのアクセスが含まれるようになりました。

ステップ 1: (オプション) AWS IoT Events ディテクターモデル設定をエクスポートする

新しいリソースを作成する前に、 AWS IoT Events ディテクターモデル定義をエクスポートします。これらにはイベント処理ロジックが含まれており、新しいソリューションを実装するための過去のリファレンスとして使用できます。

Console

を使用して AWS IoT Events AWS Management Console、次の手順を実行してディテクターモデル設定をエクスポートします。

を使用してディテクターモデルをエクスポートするには AWS Management Console

1. AWS IoT Events コンソール にログインします。
2. 左のナビゲーションペインで、[Detector models (ディテクターモデル)] を選択します。
3. エクスポートするディテクターモデルを選択します。
4. [エクスポート] を選択します。出力に関する情報メッセージを読み、もう一度エクスポートを選択します。
5. エクスポートするディテクターモデルごとにプロセスを繰り返します。

ディテクターモデルの JSON 出力を含むファイルがブラウザのダウンロードフォルダに追加されます。オプションで各ディテクターモデル設定を保存して、履歴データを保持できます。

AWS CLI

を使用して AWS CLI、次のコマンドを実行してディテクターモデル設定をエクスポートします。

を使用してディテクターモデルをエクスポートするには AWS CLI

1. アカウント内のすべてのディテクターモデルを一覧表示します。

```
aws iotevents list-detector-models
```

2. ディテクターモデルごとに、以下を実行して設定をエクスポートします。

```
aws iotevents describe-detector-model \
   --detector-model-name your-detector-model-name
```

3. 各ディテクターモデルの出力を保存します。

ステップ 2: IAM ロールを作成する

IAM ロールを作成して、 の機能をレプリケートするアクセス許可を付与します AWS IoT Events。この例のロールは、状態管理用の DynamoDB、スケジューリング用の EventBridge、データ取り込み用の Kinesis Data Streams、メッセージの発行 AWS IoT Core 用の 、ログ記録用の CloudWatch

へのアクセスを許可します。これらのサービスは、 の代替として連携して機能します AWS IoT Events。

1. 以下のアクセス許可を持つ IAM ロールを作成します。IAM ロールの作成の詳細については、IAM ユーザーガイドの「 AWS サービスにアクセス許可を委任するロールを作成する」を参照してください。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "DynamoDBAccess",
            "Effect": "Allow",
            "Action": [
                "dynamodb:GetItem",
                "dynamodb:PutItem",
                "dynamodb:UpdateItem",
                "dynamodb:DeleteItem",
                "dynamodb:Query",
                "dynamodb:Scan"
            ],
            "Resource": "arn:aws:dynamodb:your-region:your-account-id:table/
EventsStateTable"
        },
        {
            "Sid": "SchedulerAccess",
            "Effect": "Allow",
            "Action": [
                "scheduler:CreateSchedule",
                "scheduler:DeleteSchedule"
            ],
            "Resource": "arn:aws:scheduler:your-region:your-account-id:schedule/*"
        },
        {
            "Sid": "KinesisAccess",
            "Effect": "Allow",
            "Action": [
                "kinesis:GetRecords",
                "kinesis:GetShardIterator",
                "kinesis:DescribeStream",
                "kinesis:ListStreams"
            ],
            "Resource": "arn:aws:kinesis:your-region:your-account-id:stream/*"
```

```
        },
        {
            "Sid": "IoTPublishAccess",
            "Effect": "Allow",
            "Action": "iot:Publish",
            "Resource": "arn:aws:iot:your-region:your-account-id:topic/*"
        },
        {
            "Effect": "Allow",
            "Action": "logs:CreateLogGroup",
            "Resource": "arn:aws:logs:your-region:your-account-id:*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogStream",
                "logs:PutLogEvents"
            ],
            "Resource": [
                "arn:aws:logs::your-account-id:log-group:/aws/lambda/your-lambda:*"
            ]
        }
    ]
}
```

2. 次の IAM ロールの信頼ポリシーを追加します。信頼ポリシーは、指定された AWS サービスが必要なアクションを実行できるように、IAM ロールを引き受けることを許可します。IAM 信頼ポリシーの作成方法の詳細については、IAM ユーザーガイドのカスタム信頼ポリシーを使用してロールを作成するを参照してください。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": [
                    "scheduler.amazonaws.com",
                    "lambda.amazonaws.com",
                    "iot.amazonaws.com"
                ]
            },
            "Action": "sts:AssumeRole"
```

```
        }
    ]
}
```

ステップ 3: Amazon Kinesis Data Streams を作成する

AWS Management Console または を使用して Amazon Kinesis Data Streams を作成します AWS CLI。

Console

を使用して Kinesis データストリームを作成するには AWS Management Console、Amazon Kinesis Data Streams デベロッパーガイド」の「データストリームの作成」ページにある手順に従ってください。

デバイス数とメッセージペイロードサイズに基づいてシャード数を調整します。

AWS CLI

を使用して AWS CLI Amazon Kinesis Data Streams を作成し、デバイスからデータを取り込んでパーティション化します。

Kinesis Data Streams は、この移行でデータの取り込み機能を置き換えるために使用されます AWS IoT Events。IoT IoT デバイスからリアルタイムのストリーミングデータを収集、処理、分析するスケーラブルで効率的な方法を提供すると同時に、柔軟なデータ処理と他の AWS サービスとの統合を提供します。

```
aws kinesis create-stream --stream-name your-kinesis-stream-name --shard-count 4 --
region your-region
```

デバイス数とメッセージペイロードサイズに基づいてシャード数を調整します。

ステップ 4: MQTT メッセージルーティングルールを作成または更新する

新しい MQTT メッセージルーティングルールを作成するか、既存のルールを更新できます。

Console

1. 新しい MQTT メッセージルーティングルールが必要かどうか、または既存のルールを更新できるかどうかを決定します。
2. AWS IoT Core コンソールを開きます。
3. ナビゲーションペインで、メッセージルーティングを選択し、ルールを選択します。
4. 管理セクションで、メッセージルーティングを選択し、ルールを選択します。
5. [ルールを作成] を選択します。
6. ルールプロパティの指定 ページで、 AWS IoT Core ルール名のルール名を入力します。ルールの説明 - オプションで、イベントを処理して Kinesis Data Streams に転送していることを識別する説明を入力します。
7. SQL ステートメントの設定ページで、SQL ステートメントに「」と入力し**`SELECT * FROM 'your-database'`**、次へを選択します。
8. ルールアクションのアタッチページで、ルールアクションで kinesis を選択します。
9. ストリームの Kinesis ストリームを選択します。パーティションキーに「**`your-instance-id`**」と入力します。IAM ロールに適したロールを選択し、ルールの追加アクションを選択します。

詳細については、「Creating AWS IoT rules to route device data to other services」を参照してください。

AWS CLI

1. 次の内容を含む JSON ファイル を作成します。この JSON 設定ファイルは、トピックからすべてのメッセージを選択し、インスタンス ID をパーティションキーとして使用して、指定された Kinesis ストリームに転送する AWS IoT Core ルールを定義します。

```
{
    "sql": "SELECT * FROM 'your-config-file'",
    "description": "Rule to process events and forward to Kinesis Data Streams",
    "actions": [
        {
            "kinesis": {
                "streamName": "your-kinesis-stream-name",
                "roleArn": "arn:aws:iam::your-account-id:role/service-role/your-
iam-role",
                "partitionKey": "${your-instance-id}"
            }
```

```
        }
    ],
    "ruleDisabled": false,
    "awsIotSqlVersion": "2016-03-23"
}
```

2. を使用して MQTT トピックルールを作成します AWS CLI。このステップでは AWS CLI 、を使用して、 `events_rule.json` ファイルで定義された設定を使用して AWS IoT Core トピックルールを作成します。

```
aws iot create-topic-rule \
    --rule-name "your-iot-core-rule" \
    --topic-rule-payload file://your-file-name.json
```

ステップ 5: 送信先 MQTT トピックのエンドポイントを取得する

送信先 MQTT トピックを使用して、トピックが送信メッセージを発行する場所を設定し、以前に処理された機能を置き換えます AWS IoT Events。エンドポイントは、 AWS アカウントとリージョンに固有です。

Console

1. AWS IoT Core コンソールを開きます。
2. 左側のナビゲーションパネルの接続セクションで、ドメイン設定を選択します。
3. iot:Data-ATS ドメイン設定を選択して、設定の詳細ページを開きます。
4. ドメイン名の値をコピーします。この値はエンドポイントです。後のステップで必要になるため、エンドポイント値を保存します。

AWS CLI

次のコマンドを実行して、アカウントの送信メッセージを発行するための AWS IoT Core エンドポイントを取得します。

```
aws iot describe-endpoint --endpoint-type iot:Data-ATS --region your-region
```

ステップ 6: Amazon DynamoDB テーブルを作成する

Amazon DynamoDB テーブルは、 の状態管理機能を置き換え AWS IoT Events、新しいソリューションアーキテクチャでデバイスの状態とディテクターモデルロジックを持続および管理するためのスケーラブルで柔軟な方法を提供します。

Console

Amazon DynamoDB テーブルを作成して、ディテクターモデルの状態を維持します。詳細については、「Amazon DynamoDB デベロッパーガイド」の「DynamoDB でテーブルを作成する」を参照してください。 DynamoDB

テーブルの詳細については、以下を使用します。

- テーブル名には、選択したテーブル名を入力します。
- パーティションキーには、独自のインスタンス ID を入力します。
- テーブル設定のデフォルト設定を使用できます。

AWS CLI

次のコマンドを実行して、DynamoDB テーブルを作成します。

```
aws dynamodb create-table \
                        --table-name your-table-name \
                        --attribute-definitions AttributeName=your-instance-
id,AttributeType=S \
                        --key-schema AttributeName=your-instance-id,KeyType=HASH \
```

ステップ 7: AWS Lambda 関数を作成する (コンソール)

Lambda 関数はコア処理エンジンとして機能し、ディテクターモデルの評価ロジックを置き換えます AWS IoT Events。この例では、他の AWS サービスと統合して、定義されたルールに基づいて受信データを処理し、状態を管理し、アクションをトリガーします。

NodeJS ランタイムを使用して Lambda 関数を作成します。次のコードスニペットを使用して、ハードコードされた定数を置き換えます。

1. AWS Lambda consoleを開きます。
2. [関数の作成] を選択してください。

3. 関数名の名前を入力します。
4. ランタイムとして NodeJS 22.x を選択します。
5. デフォルトの実行ロールの変更ドロップダウンで、既存のロールを使用を選択し、前のステップで作成した IAM ロールを選択します。
6. [関数の作成] を選択してください。
7. ハードコードされた定数を置き換えた後、次のコードスニペットに貼り付けます。
8. 関数を作成したら、Code タブに次のコード例を貼り付け、**your-destination-endpoint**エンドポイントを独自のコードに置き換えます。

```
import { DynamoDBClient, GetItemCommand } from '@aws-sdk/client-dynamodb';
import { PutItemCommand } from '@aws-sdk/client-dynamodb';
import { IoTDataPlaneClient, PublishCommand } from "@aws-sdk/client-iot-data-plane";
import { SchedulerClient, CreateScheduleCommand, DeleteScheduleCommand } from "@aws-
sdk/client-scheduler"; // ES Modules import


//// External Clients and Constants
const scheduler = new SchedulerClient({});
const iot = new IoTDataPlaneClient({
    endpoint: 'https://your-destination-endpoint-ats.iot.your-region.amazonaws.com/'
});
const ddb = new DynamoDBClient({});


//// Lambda Handler function
export const handler = async (event) => {
    console.log('Incoming event:', JSON.stringify(event, null, 2));

    if (!event.Records) {
        throw new Error('No records found in event');
    }

    const processedRecords = [];

    for (const record of event.Records) {
        try {
            if (record.eventSource !== 'aws:kinesis') {
                console.log(`Skipping non-Kinesis record from ${record.eventSource}`);
                continue;
```

```
            }

            // Assumes that we are processing records from Kinesis
            const payload = record.kinesis.data;
            const decodedData = Buffer.from(payload, 'base64').toString();
            console.log("decoded payload is ", decodedData);

            const output = await handleDecodedData(decodedData);

            // Add additional processing logic here
            const processedData = {
                output,
                sequenceNumber: record.kinesis.sequenceNumber,
                partitionKey: record.kinesis.partitionKey,
                timestamp: record.kinesis.approximateArrivalTimestamp
            };

            processedRecords.push(processedData);

        } catch (error) {
            console.error('Error processing record:', error);
            console.error('Failed record:', record);
            // Decide whether to throw error or continue processing other records
            // throw error; // Uncomment to stop processing on first error
        }
    }

    return {
        statusCode: 200,
        body: JSON.stringify({
            message: 'Processing complete',
            processedCount: processedRecords.length,
            records: processedRecords
        })
    };
};

// Helper function to handle decoded data
async function handleDecodedData(payload) {
    try {
        // Parse the decoded data
        const parsedData = JSON.parse(payload);

        // Extract instanceId
```

```
        const instanceId = parsedData.instanceId;
        // Parse the input field
        const inputData = JSON.parse(parsedData.payload);
        const temperature = inputData.temperature;
        console.log('For InstanceId: ', instanceId, ' the temperature is:',
 temperature);

        await iotEvents.process(instanceId, inputData)

        return {
            instanceId,
            temperature,
            // Add any other fields you want to return
            rawInput: inputData
        };
    } catch (error) {
        console.error('Error handling decoded data:', error);
        throw error;
    }
}


//// Classes for declaring/defining the state machine
class CurrentState {
    constructor(instanceId, stateName, variables, inputs) {
        this.stateName = stateName;
        this.variables = variables;
        this.inputs = inputs;
        this.instanceId = instanceId
    }

    static async load(instanceId) {
        console.log(`Loading state for id ${instanceId}`);
        try {
            const { Item: { state: { S: stateContent } } } = await ddb.send(new
 GetItemCommand({
                TableName: 'EventsStateTable',
                Key: {
                    'InstanceId': { S: `${instanceId}` }
                }
            }));

            const { stateName, variables, inputs } = JSON.parse(stateContent);
```

```
            return new CurrentState(instanceId, stateName, variables, inputs);
        } catch (e) {
            console.log(`No state for id ${instanceId}: ${e}`);
            return undefined;
        }
    }

    static async save(instanceId, state) {
        console.log(`Saving state for id ${instanceId}`);
        await ddb.send(new PutItemCommand({
            TableName: 'your-events-state-table-name',
            Item: {
                'InstanceId': { S: `${instanceId}` },
                'state': { S: state }
            }
        }));
    }

    setVariable(name, value) {
        this.variables[name] = value;
    }

    changeState(stateName) {
        console.log(`Changing state from ${this.stateName} to ${stateName}`);
        this.stateName = stateName;
    }

    async setTimer(instanceId, frequencyInMinutes, payload) {
        console.log(`Setting timer ${instanceId} for frequency of ${frequencyInMinutes}
 minutes`);

        const base64Payload = Buffer.from(JSON.stringify(payload)).toString();
        console.log(base64Payload);

        const scheduleName = `your-schedule-name-${instanceId}-schedule`;
        const scheduleParams = {
            Name: scheduleName,
            FlexibleTimeWindow: {
                Mode: 'OFF'
            },
            ScheduleExpression: `rate(${frequencyInMinutes} minutes)`,
            Target: {
                Arn: "arn:aws::kinesis:your-region:your-account-id:stream/your-kinesis-
stream-name",
```

```
                RoleArn: "arn:aws::iam::your-account-id:role/service-role/your-iam-
role",
                Input: base64Payload,
                KinesisParameters: {
                    PartitionKey: instanceId,
                },
                RetryPolicy: {
                    MaximumRetryAttempts: 3
                }
            },

        };

        const command = new CreateScheduleCommand(scheduleParams);
        console.log(`Sending command to set timer ${JSON.stringify(command)}`);
        await scheduler.send(command);
    }

    async clearTimer(instanceId) {
        console.log(`Cleaning timer ${instanceId}`);

        const scheduleName = `your-schedule-name-${instanceId}-schdule`;
        const command = new DeleteScheduleCommand({
            Name: scheduleName
        });
        await scheduler.send(command);
    }

    async executeAction(actionType, actionPayload) {
        console.log(`Will execute the ${actionType} with payload ${actionPayload}`);
        await iot.send(new PublishCommand({
            topic: `${this.instanceId}`,
            payload: actionPayload,
            qos: 0
        }));
    }

    setInput(value) {
        this.inputs = { ...this.inputs, ...value };
    }

    input(name) {
        return this.inputs[name];
    }
```

```
}


class IoTEvents {

    constructor(initialState) {
        this.initialState = initialState;
        this.states = {};
    }

    state(name) {
        const state = new IoTEventsState();
        this.states[name] = state;
        return state;
    }

    async process(instanceId, input) {
        let currentState = await CurrentState.load(instanceId) || new
 CurrentState(instanceId, this.initialState, {}, {});
        currentState.setInput(input);

        console.log(`With inputs as: ${JSON.stringify(currentState)}`);
        const state = this.states[currentState.stateName];

        currentState = await state.evaluate(currentState);
        console.log(`With output as: ${JSON.stringify(currentState)}`);

        await CurrentState.save(instanceId, JSON.stringify(currentState));
    }
}

class Event {
    constructor(condition, action) {
        this.condition = condition;
        this.action = action;
    }
}

class IoTEventsState {
    constructor() {
        this.eventsList = []
    }

    events(eventListArg) {
```

```
        this.eventsList.push(...eventListArg);
        return this;
    }

    async evaluate(currentState) {
        for (const e of this.eventsList) {
            console.log(`Evaluating event ${e.condition}`);
            if (e.condition(currentState)) {
                console.log(`Event condition met`);
                // Execute any action as defined in iotEvents DM Definition
                await e.action(currentState);
            }
        }

        return currentState;
    }
}

////// DetectorModel Definitions - replace with your own defintions
let processAlarmStateEvent = new Event(
    (currentState) => {
        const source = currentState.input('source');
        return (
            currentState.input('temperature') < 70
        );
    },
    async (currentState) => {
        currentState.changeState('normal');
        await currentState.clearTimer(currentState.instanceId)
        await currentState.executeAction('MQTT', `{"state": "alarm cleared, timer
 deleted" }`);
    }
);

let processTimerEvent = new Event(
    (currentState) => {
        const source = currentState.input('source');
        console.log(`Evaluating timer event with source ${source}`);
        const booleanOutput = (source !== undefined && source !== null &&
            typeof source === 'string' &&
            source.toLowerCase() === 'timer' &&
            // check if the currentState == state from the timer payload
            currentState.input('currentState') !== undefined &&
            currentState.input('currentState') !== null &&
```

```
            currentState.input('currentState').toLowerCase !== 'normal');
        console.log(`Timer event evaluated as ${booleanOutput}`);
        return booleanOutput;
    },
    async (currentState) => {
        await currentState.executeAction('MQTT', `{"state": "timer timed out in
 Alarming state" }`);
    }
);

let processNormalEvent = new Event(
    (currentState) => currentState.input('temperature') > 70,
    async (currentState) => {
        currentState.changeState('alarm');
        await currentState.executeAction('MQTT', `{"state": "alarm detected, timer
 started" }`);
        await currentState.setTimer(currentState.instanceId, 5, {
            "instanceId": currentState.instanceId,
            "payload":"{\"currentState\": \"alarm\", \"source\": \"timer\"}"
        });
    }
);
const iotEvents = new IoTEvents('normal');
iotEvents
    .state('normal')
    .events(
        [
            processNormalEvent
        ]);
iotEvents
    .state('alarm')
    .events([
            processAlarmStateEvent,
            processTimerEvent
        ]
    );
```

ステップ 8: Amazon Kinesis Data Streams トリガーを追加する

AWS Management Console または を使用して、Kinesis Data Streams トリガーを Lambda 関数に追加します AWS CLI。

Lambda 関数に Kinesis Data Streams トリガーを追加すると、データ取り込みパイプラインと処理ロジック間の接続が確立され、入力 AWS IoT Events の処理方法と同様に、受信 IoT データストリームを自動的に評価し、イベントにリアルタイムで対応できるようになります。

Console

詳細については、「 デベロッパーガイド」の「Lambda 関数を呼び出すイベントソースマッピングを作成する」を参照してください。 AWS Lambda

イベントソースマッピングの詳細には、以下を使用します。

- 関数名には、 で使用される Lambda 名を入力しますステップ 7: AWS Lambda 関数を作成する (コンソール)。
- コンシューマー - オプションで、Kinesis ストリームの ARN を入力します。
- [バッチサイズ] に、**10** を入力します。

AWS CLI

次のコマンドを実行して、Lambda 関数トリガーを作成します。

```
aws lambda create-event-source-mapping \
    --function-name your-lambda-name \
    --event-source arn:aws:kinesis:your-region:your-account-id:stream/your-kinesis-
stream-name \
    --batch-size 10 \
    --starting-position LATEST \
    --region your-region
```

ステップ 9: データの取り込みと出力機能をテストする (AWS CLI)

ディテクターモデルで定義した内容に基づいて MQTT トピックにペイロードを発行します。実装をテストyour-topic-nameするための MQTT トピックへのペイロードの例を次に示します。

```
{
  "instanceId": "your-instance-id",
  "payload": "{\"temperature\":78}"
}
```

次の (または同様の) コンテンツを含むトピックに発行された MQTT メッセージが表示されます。

```
{
    "state": "alarm detected, timer started"
}
```

での AWS IoT SiteWise アラームの移行手順 AWS IoT Events

このセクションでは、移行時と同様のアラーム機能を提供する代替ソリューションについて説明します AWS IoT Events。

AWS IoT Events アラームを使用する AWS IoT SiteWise プロパティの場合、CloudWatch アラームを使用してソリューションに移行できます。このアプローチは、確立された SLAs と、異常検出やグループ化されたアラームなどの追加機能を備えた堅牢なモニタリング機能を提供します。

アーキテクチャの比較

AWS IoT SiteWise プロパティの現在の AWS IoT Events アラーム設定では、AWS IoT SiteWise 「ユーザーガイドAssetModelCompositeModels」の「で外部アラームを定義する AWS IoT SiteWise」で説明されているように、アセットモデルで を作成する必要があります。新しいソリューションへの変更は、通常、 コンソールを通じて管理されます AWS IoT Events 。

新しいソリューションは、CloudWatch アラームを活用してアラーム管理を提供します。このアプローチでは、 AWS IoT SiteWise 通知を使用してプロパティデータポイントを AWS IoT Core MQTT トピックに発行し、Lambda 関数によって処理されます。この関数は、これらの通知を CloudWatch メトリクスに変換し、CloudWatch のアラームフレームワークによるアラームモニタリングを有効にします。

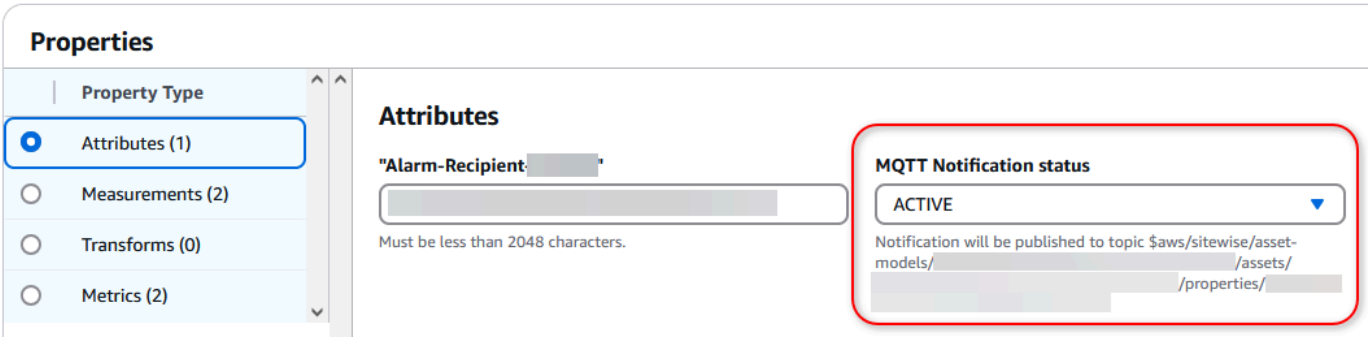
目的	ソリューション	相違点
データソース – からのプロパティデータ AWS IoT SiteWise	AWS IoT SiteWise MQTT 通知	IoT IoT Events の直接統合を AWS IoT SiteWise プロパティからの MQTT 通知に置き換えます
データ処理 – プロパティデータを変換します	Lambda function	AWS IoT SiteWise プロパティ通知を処理し、CloudWatch メトリクスに変換します
アラーム評価 – メトリクスをモニタリン	Amazon CloudWatch アラーム	AWS IoT Events アラームを CloudWatch アラームに置き換え、異常検出などの追加機能を提供します

目的	ソリューション	相違点
グし、アラームをトリガーします		
統合 – との接続 AWS IoT SiteWise	AWS IoT SiteWise 外部アラーム	CloudWatch アラームを外部アラーム AWS IoT SiteWise として にインポートするオプション機能

ステップ 1: アセットプロパティで MQTT 通知を有効にする

AWS IoT SiteWise アラーム AWS IoT Events の統合を使用している場合は、モニタリングする各プロパティの MQTT 通知を有効にできます。

1. 各ステップでアセットモデルのプロパティを編集するまで、「Configure alarm on assets in AWS IoT SiteWise procedure」に従います。
2. 移行するプロパティごとに、MQTT 通知のステータスを ACTIVE に変更します。



3. 変更されたアラーム属性ごとにアラームが発行するトピックパスを書き留めます。

詳細については、以下のドキュメントリソースを参照してください。

- AWS IoT SiteWise ユーザーガイドの MQTT トピックのアセットプロパティについて説明します。
- AWS IoT デベロッパーガイドの MQTT トピック。

ステップ 2: AWS Lambda 関数を作成する

MQTT トピックによって発行された TQV 配列を読み取るための Lambda 関数を作成し、個々の値を CloudWatch に発行します。メッセージ AWS IoT Core ルールでトリガーする送信先アクションとして、この Lambda 関数を使用します。

1. AWS Lambda consoleを開きます。
2. [関数の作成] を選択してください。
3. 関数名の名前を入力します。
4. ランタイムとして NodeJS 22.x を選択します。
5. デフォルトの実行ロールの変更ドロップダウンで、既存のロールを使用を選択し、前のステップで作成した IAM ロールを選択します。

> **Note**
>
> この手順では、ディテクターモデルが既に移行されていることを前提としています。IAM ロールがない場合は、「」を参照してください???。

6. [関数の作成] を選択してください。
7. ハードコードされた定数を置き換えた後、次のコードスニペットに貼り付けます。

```
import json
import boto3
from datetime import datetime

# Initialize CloudWatch client
cloudwatch = boto3.client('cloudwatch')

def lambda_handler(message, context):
    try:
        # Parse the incoming IoT message
        # Extract relevant information
        asset_id = message['payload']['assetId']
        property_id = message['payload']['propertyId']

        # Process each value in the values array
        for value in message['payload']['values']:
            # Extract timestamp and value
            timestamp = datetime.fromtimestamp(value['timestamp']['timeInSeconds'])
            metric_value = value['value']['doubleValue']
```

```
            quality = value.get('quality', 'UNKNOWN')

            # Publish to CloudWatch
            response = cloudwatch.put_metric_data(
                Namespace='IoTSiteWise/AssetMetrics',
                MetricData=[
                    {
                        'MetricName': f'Property_your-property-id',
                        'Value': metric_value,
                        'Timestamp': timestamp,
                        'Dimensions': [
                            {
                                'Name': 'AssetId',
                                'Value': 'your-asset-id'
                            },
                            {
                                'Name': 'Quality',
                                'Value': quality
                            }
                        ]
                    }
                ]
            )

        return {
            'statusCode': 200,
            'body': json.dumps('Successfully published metrics to CloudWatch')
        }

    except Exception as e:
        print(f'Error processing message: {str(e)}')
        return {
            'statusCode': 500,
            'body': json.dumps(f'Error: {str(e)}')
        }
```

ステップ 3: AWS IoT Core メッセージルーティングルールを作成する

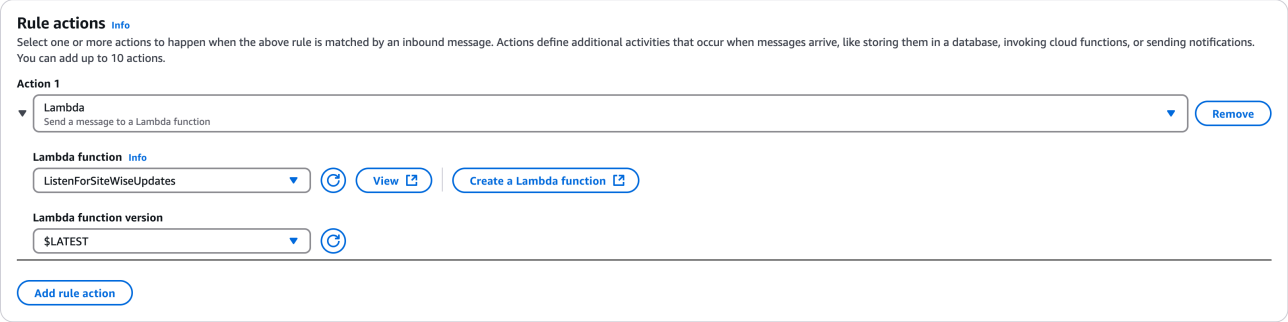
- チュートリアル: MQTT メッセージの再発行手順に従い、プロンプトが表示されたら次の情報を入力します。

  a. 名前メッセージルーティングルール `SiteWiseToCloudwatchAlarms`。

b. クエリには、以下を使用できます。

```
SELECT * FROM '$aws/sitewise/asset-models/your-asset-model-id/assets/your-asset-id/properties/your-property-id'
```

c. ルールアクションで、 から生成されたデータを AWS IoT SiteWise CloudWatch に送信する Lambda アクションを選択します。例:



ステップ 4: CloudWatch メトリクスを表示する

で前に選択した プロパティにデータを取り込むと AWS IoT SiteWise、 は ステップ 1: アセットプロパティで MQTT 通知を有効にするで作成した Lambda 関数にデータをルーティングしますステップ 2: AWS Lambda 関数を作成する。このステップでは、Lambda がメトリクスを CloudWatch に送信していることを確認できます。

1. CloudWatch AWS Management Consoleを開きます。
2. 左側のナビゲーションで、メトリクスを選択し、次にすべてのメトリクスを選択します。
3. アラームの URL を選択して開きます。
4. ソースタブの CloudWatch 出力は次の例のようになります。このソース情報は、メトリクスデータが CloudWatch にフィードされていることを確認します。

```
{
    "view": "timeSeries",
    "stacked": false,
    "metrics": [
        [ "IoTSiteWise/AssetMetrics", "Property_your-property-id-hash", "Quality",
 "GOOD", "AssetId", "your-asset-id-hash", { "id": "m1" } ]
    ],
    "region": "your-region"
}
```

ステップ 5: CloudWatch アラームを作成する

Amazon CloudWatch ユーザーガイドの静的しきい値に基づく CloudWatch アラームの作成手順に従って、関連する各メトリクスのアラームを作成します。 Amazon CloudWatch

> **Note**
>
> Amazon CloudWatch のアラーム設定には多くのオプションがありますCloudWatch アラームの詳細については、Amazon CloudWatch ユーザーガイド」の「Amazon CloudWatch アラームの使用」を参照してください。 Amazon CloudWatch

ステップ 6: (オプション) CloudWatch アラームを にインポートする AWS IoT SiteWise

CloudWatch アラームアクションと Lambda AWS IoT SiteWise を使用してデータを に送信するように CloudWatch アラームを設定できます。この統合により、SiteWise Monitor ポータルでアラームの状態とプロパティを表示できます。

1. 外部アラームをアセットモデルのプロパティとして設定します。詳細については、「 AWS IoT SiteWise ユーザーガイド」の「 で外部アラームを定義する AWS IoT SiteWise」を参照してください。
2. AWS IoT SiteWise ユーザーガイドにある BatchPutAssetPropertyValue API を使用してアラームデータを に送信する Lambda 関数を作成します AWS IoT SiteWise。
3. アラーム状態が変更されたときに Lambda 関数を呼び出すように CloudWatch アラームアクションを設定します。詳細については、Amazon CloudWatch ユーザーガイド」の「アラームアクション」セクションを参照してください。

セットアップ AWS IoT Events

このセクションでは、 AWS アカウントの作成 AWS IoT Events、必要なアクセス許可の設定、 リソースへのアクセスを管理するためのロールの確立など、 を設定するためのガイドを提供します。

トピック

- のセットアップ AWS アカウント
- のアクセス許可の設定 AWS IoT Events

のセットアップ AWS アカウント

にサインアップする AWS アカウント

がない場合は AWS アカウント、次の手順を実行して作成します。

にサインアップするには AWS アカウント

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

   サインアップ手順の一環として、電話またはテキストメッセージを受け取り、電話キーパッドで検証コードを入力します。

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

AWS サインアッププロセスが完了すると、 から確認メールが送信されます。https://aws.amazon.com/ の [マイアカウント] をクリックして、いつでもアカウントの現在のアクティビティを表示し、アカウントを管理することができます。

管理アクセスを持つユーザーを作成する

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

を保護する AWS アカウントのルートユーザー

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

   ルートユーザーを使用してサインインする方法については、AWS サインイン ユーザーガイドのルートユーザーとしてサインインするを参照してください。

2. ルートユーザーの多要素認証 (MFA) を有効にします。

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

管理アクセスを持つユーザーを作成する

1. IAM アイデンティティセンターを有効にします。

   手順については、「AWS IAM Identity Center ユーザーガイド」の「AWS IAM Identity Centerの有効化」を参照してください。

2. IAM アイデンティティセンターで、ユーザーに管理アクセスを付与します。

   を ID ソース IAM アイデンティティセンターディレクトリ として使用する方法のチュートリアルについては、「 AWS IAM Identity Center ユーザーガイド」の「デフォルトを使用してユーザーアクセスを設定する IAM アイデンティティセンターディレクトリ」を参照してください。

管理アクセス権を持つユーザーとしてサインインする

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

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

追加のユーザーにアクセス権を割り当てる

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

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

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

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

のアクセス許可の設定 AWS IoT Events

の安全で効果的な使用には、適切なアクセス許可を実装することが重要です AWS IoT Events。このセクションでは、 の一部の機能を使用するために必要なアクセス許可について説明します AWS IoT Events。 AWS CLI コマンドまたは AWS Identity and Access Management (IAM) コンソールを使用して、 リソースにアクセスしたり、 で特定の機能を実行したりするためのロールおよび関連するアクセス許可ポリシーを作成できます AWS IoT Events。

IAM ユーザーガイドには、 AWS リソースへのアクセス許可の安全な制御に関する詳細情報が記載されています。固有の情報については AWS IoT Events、「 のアクション、リソース、および条件キー AWS IoT Events」を参照してください。

IAM コンソールを使用してロールとアクセス許可を作成および管理するには、「IAM チュートリアル: IAM ロールを使用して AWS アカウント間のアクセスを委任する」を参照してください。

> **Note**
>
> キーは 1 ~ 128 文字の長さで、以下を含めることができます。
>
> - 大文字または小文字 (a ~ z)
> - 数字 (0〜9)
> - 特殊文字: -、_、:

のアクションアクセス許可 AWS IoT Events

AWS IoT Events では、他の AWS サービスを使用するアクションをトリガーできます。そのためには、ユーザーに代わってこれらのアクションを実行する AWS IoT Events アクセス許可を付与する必要があります。このセクションには、アクション一覧と、これらのアクションをお客様のリソースに対して実行する許可を与えるポリシーの例が含まれています。必要に応じて `#####` と `account-id` の参照を変更します。可能であれば、アクセスされる特定のリソースを参照するようにワイルドカード (*) も変更する必要があります。IAM コンソールを使用して、定義した Amazon SNS アラートを送信するアクセス許可 AWS IoT Events を に付与できます。

AWS IoT Events では、タイマーを使用したり、変数を設定したりできる以下のアクションがサポートされています。

- `setTimer` は、タイマーを作成します。
- `resetTimer` は、タイマーをリセットします。
- `clearTimer` は、タイマーを削除します。
- `setVariable` は、可変を作成します。

AWS IoT Events では、 AWS サービスを操作できる以下のアクションがサポートされています。

- `iotTopicPublish` は、MQTT トピックにメッセージを発行します。
- `iotEvents` は、 AWS IoT Events に対してデータを入力値として送信します。
- `iotSiteWise` は、 AWS IoT SiteWise中のアセットプロパティにデータを送信します。
- `dynamoDB` は、Amazon DynamoDB テーブルにデータを送信します。
- `dynamoDBv2` は、Amazon DynamoDB テーブルにデータを送信します。
- `firehose` は Amazon Data Firehose ストリームにデータを送信します。
- `lambda` は、 AWS Lambda 関数を呼び出します。
- `sns` は、データをプッシュ通知として送信します。
- `sqs` は、Amazon SQS キューにデータを送信します。

Example ポリシー

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

```
        "Effect": "Allow",
        "Action": "iot:Publish",
        "Resource": "arn:aws:iot:<region>:<account_id>:topic/*"
      },
      {
        "Effect": "Allow",
        "Action": "iotevents:BatchPutMessage",
        "Resource": "arn:aws:iotevents:<region>:<account_id>:input/*"
      },
      {
        "Effect": "Allow",
        "Action": "iotsitewise:BatchPutAssetPropertyValue",
        "Resource": "*"
      },
      {
        "Effect": "Allow",
        "Action": "dynamodb:PutItem",
        "Resource": "arn:aws:dynamodb:<region>:<account_id>:table/*"
      },
      {
        "Effect": "Allow",
        "Action": [
          "firehose:PutRecord",
          "firehose:PutRecordBatch"
        ],
        "Resource": "arn:aws:firehose:<region>:<account_id>:deliverystream/*"
      },
      {
        "Effect": "Allow",
        "Action": "lambda:InvokeFunction",
        "Resource": "arn:aws:lambda:<region>:<account_id>:function:*"
      },
      {
        "Effect": "Allow",
        "Action": "sns:Publish",
        "Resource": "arn:aws:sns:<region>:<account_id>:*"
      },
      {
        "Effect": "Allow",
        "Action": "sqs:SendMessage",
        "Resource": "arn:aws:sqs:<region>:<account_id>:*"
      }
    ]
```

```
}
```

での入力データの保護 AWS IoT Events

ディテクタモデルで使用する入力データへのアクセスを誰に許可できるかを検討することが重要です。全体的な許可を制限したいが、ディテクタモデルの作成または更新が許可されているユーザーまたはエンティティがある場合は、そのユーザーまたはエンティティに入力ルーティングを更新する許可も付与する必要があります。つまり、`iotevents:CreateDetectorModel` と `iotevents:UpdateDetectorModel` の許可に加えて、`iotevents:UpdateInputRouting` の許可も付与する必要があります。

Example

次のポリシーは、`iotevents:UpdateInputRouting` の許可を追加します。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "updateRoutingPolicy",
            "Effect": "Allow",
            "Action": [
                "iotevents:UpdateInputRouting"
            ],
            "Resource": "*"
        }
    ]
}
```

「`Resource`」のワイルドカード「`*`」の代わりに、入力 Amazon リソースネーム (ARN) のリストを指定して、この許可を特定の入力に許可することができます。これにより、このユーザーまたはエンティティによって作成または更新されたディテクタモデルが消費する入力データに限定してアクセス許可を与えることができます。

の Amazon CloudWatch ログ記録ロールポリシー AWS IoT Events

以下のポリシードキュメントは、 AWS IoT Events がお客様に代わって CloudWatch にログを送信できるようにするロールポリシーと信頼ポリシーを提供します。

ロールポリシー:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogGroup",
                "logs:CreateLogStream",
                "logs:PutLogEvents",
                "logs:PutMetricFilter",
                "logs:PutRetentionPolicy",
                "logs:GetLogEvents",
                "logs:DeleteLogStream"
            ],
            "Resource": [
                "arn:aws:logs:*:*:*"
            ]
        }
    ]
}
```

信頼ポリシー:

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

          "iotevents.amazonaws.com"
        ]
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

また、次のようにして、IAM のアクセス許可ポリシーをユーザーにアタッチして、ユーザーがロールを渡せるようにする必要もあります。詳細については、IAM ユーザーガイドの「 AWS サービスにロールを渡すアクセス許可をユーザーに付与する」を参照してください。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "",
      "Effect": "Allow",
      "Action": [
          "iam:GetRole",
          "iam:PassRole"
      ],
      "Resource": "arn:aws:iam::<account-id>:role/Role_To_Pass"
    }
  ]
}
```

次のコマンドを使用して、CloudWatch Logs に対するリソースポリシーを設定できます。これにより、 AWS IoT Events が CloudWatch ストリーミングにログイベントを配置することが許可になります。

```
aws logs put-resource-policy --policy-name ioteventsLoggingPolicy --policy-
document "{ \"Version\": \"2012-10-17\", \"Statement\": [ { \"Sid\":
 \"IoTEventsToCloudWatchLogs\", \"Effect\": \"Allow\", \"Principal\": { \"Service\":
 [ \"iotevents.amazonaws.com\" ] }, \"Action\":\"logs:PutLogEvents\", \"Resource\": \"*
\" } ] }"
```

ログ記録オプションを設定するには、以下のコマンドを使用します。`roleArn` は、作成したログ記録ロールで置き換えてください。

```
aws iotevents put-logging-options --cli-input-json "{ \"loggingOptions\": {\"roleArn\":
 \"arn:aws:iam::123456789012:role/testLoggingRole\", \"level\": \"INFO\", \"enabled\":
 true } }"
```

の Amazon SNS メッセージングロールポリシー AWS IoT Events

Amazon SNS AWS IoT Events との統合には、安全で効率的な通知配信のための慎重なアクセス許可管理が必要です。このガイドでは、 が Amazon SNS トピック AWS IoT Events にメッセージを発行できるように IAM ロールとポリシーを設定するプロセスについて説明します。

以下のポリシードキュメントは、お客様に代わって AWS IoT Events から SNS メッセージを送信する許可を付与するロールポリシーと信頼ポリシーです。

ロールポリシー:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "sns:*"
            ],
            "Effect": "Allow",
            "Resource": "arn:aws:sns:us-east-1:123456789012:testAction"
        }
    ]
}
```

信頼ポリシー:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "",
      "Effect": "Allow",
      "Principal": {
        "Service": [
          "iotevents.amazonaws.com"
        ]
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

AWS IoT Events コンソールの開始方法

このセクションでは、AWS IoT Events コンソールを使用して入力モデルとディテクターモデルを作成する方法を示します。エンジンの 2 つの状態をモデル化します。通常の状態と過圧状態です。エンジン内の測定された圧力が特定のしきい値を超えると、モデルは通常の状態から過圧状態に移行します。次に、Amazon SNS メッセージを送信して、技術者に状態を警告します。圧力が 3 回連続して圧力測定値のしきい値を下回ると、モデルは通常の状態に戻り、確認として別の Amazon SNS メッセージを送信します。

非線形回復フェーズまたは異常な圧力測定値の場合に、過圧または正常メッセージの吃音の可能性を排除するために、圧力しきい値を下回る 3 つの連続した測定値をチェックします。

コンソールには、カスタマイズ可能ないくつかの既成のディテクターモデルテンプレートもあります。コンソールを使用して、他のユーザーが記述したディテクターモデルをインポートしたり、ディテクターモデルをエクスポートしたり、異なる AWS リージョンで使用したりすることもできます。ディテクターモデルをインポートする場合は、必要な入力を作成するか、新しいリージョン用にそれらを再作成し、使用されているロール ARN を更新してください。

AWS IoT Events コンソールを使用して、以下について学習します。

入力を定義する

デバイスとプロセスをモニタリングするには、テレメトリデータを AWS IoT Eventsに取り込む方法が必要です。これは、入力としてメッセージを送信することによって行われます AWS IoT Events。これはいくつかの方法で行うことができます。

- BatchPutMessageオペレーションを使用します。
- で AWS IoT Core、メッセージデータを転送するルールエンジンのAWS IoT Events アクション AWS IoT ルールを記述します AWS IoT Events。入力を名前で識別する必要があります。
- で AWS IoT Analytics、CreateDataset オペレーションを使用して、 でデータセットを作成しますcontentDeliveryRules。これらのルールは、データセットコンテンツが自動的に送信される AWS IoT Events 入力を指定します。

デバイスがこの方法でデータを送信する前に、1 つ以上の入力を定義する必要があります。このために、各入力に名前を付け、入力がモニタリングする着信メッセージデータのフィールドを指定します。

ディテクターモデルの作成

状態を使用して、ディテクターモデル (機器またはプロセスのモデル) を定義します。状態ごとに、重要なイベントを検出するために着信入力を評価する条件付き (ブール) ロジックを定義します。ディテクターモデルがイベントを検出すると、状態を変更したり、他の AWS サービスを使用してカスタムビルドまたは事前定義されたアクションを開始したりできます。状態に入るときまたは状態を出るとき、およびオプションで条件が満たされたときにアクションを開始する追加のイベントを定義できます。

このチュートリアルでは、モデルが特定の状態に出入りするときのアクションとして Amazon SNS メッセージを送信します。

デバイスまたはプロセスをモニタリングする

複数のデバイスまたはプロセスをモニタリングする場合は、入力元の特定のデバイスまたはプロセスを識別するフィールドを各入力に指定します。`CreateDetectorModel` の `key` フィールドを参照してください。`key` によって識別される入力フィールドが新しい値を認識すると、新しいデバイスが識別され、ディテクターが作成されます。各ディテクターは、ディテクターモデルのインスタンスです。新しいディテクターは、ディテクターモデルが更新または削除されるまで、そのデバイスからの入力に応答し続けます。

単一のプロセスをモニタリングする場合 (複数のデバイスまたはサブプロセスが入力を送信している場合でも)、一意の識別 `key` フィールドを指定しません。この場合、最初の入力が到着したときに、モデルは単一のディテクター (インスタンス) を作成します。

ディテクターモデルへの入力としてメッセージを送信します

デバイスまたはプロセスから AWS IoT Events ディテクターへの入力としてメッセージを送信する方法はいくつかあり、メッセージに対して追加のフォーマットを実行する必要はありません。このチュートリアルでは、 AWS IoT コンソールを使用して、メッセージデータを転送する ルールエンジンのAWS IoT Events アクション AWS IoT ルールを記述します AWS IoT Events。

これを行うには、入力を名前で識別し、引き続き AWS IoT コンソールを使用して、入力として転送されるメッセージを生成します AWS IoT Events。

> **Note**
>
> このチュートリアルでは、コンソールを使用して、AWS IoT Events ユースケースのチュートリアル の例に示されているものと同じ `input` と `detector model` を作成します。この JSON の例を使用して、チュートリアルに従うのに役立てることができます。

トピック

- の使用を開始するための前提条件 AWS IoT Events
- でモデルの入力を作成する AWS IoT Events
- でディテクターモデルを作成する AWS IoT Events
- でディテクターモデルをテストするための入力を送信する AWS IoT Events

の使用を開始するための前提条件 AWS IoT Events

AWS アカウントをお持ちでない場合は、作成してください。

1. 「」のステップに従ってセットアップ AWS IoT Events、適切なアカウント設定とアクセス許可を確保します。
2. 2 つの Amazon Simple Notification Service (Amazon SNS) トピックを作成します。

   このチュートリアル (および対応する例) は、2 つの Amazon SNS トピックを作成したことを前提としています。これらのトピックのARNは、`arn:aws:sns:us-east-1:123456789012:underPressureAction` および `arn:aws:sns:us-east-1:123456789012:pressureClearedAction` として表示されます。これらの値を、作成した Amazon SNS トピックの ARM に置き換えます。詳細については、「Amazon Simple Notification Service デベロッパーガイド」を参照してください。

   Amazon SNS トピックにアラートを公開する代わりに、指定したトピックを含む MQTT メッセージをディテクターに送信させることができます。このオプションを使用すると、 AWS IoT Core コンソールを使用してそれらの MQTT トピックに送信されたメッセージをサブスクライブおよびモニタリングすることで、ディテクターモデルがインスタンスを作成し、それらのインスタンスがアラートを送信していることを確認できます。ディテクターモデルで作成された入力または可変を使用して、ランタイムに MQTT トピック名を動的に定義することもできます。
3. がサポート AWS リージョン する を選択します AWS IoT Events。詳細については、「AWS 全般のリファレンス」の「AWS IoT Events」を参照してください。ヘルプについては、「 の開始方法」の「 のサービス AWS Management Consoleの開始方法 AWS Management Console」を参照してください。

でモデルの入力を作成する AWS IoT Events

モデルの入力を構築するときに、デバイスまたはプロセスが健全性ステータスを報告するために送信するメッセージのペイロードのサンプルを含むファイルを収集することをお勧めします。それらのファイルを用意しておくと、必要な入力を定義するのに役立ちます。

入力は、このセクションで説明する複数の方法で作成できます。

JSON 入力ファイルを作成する

1. 最初に、ローカルファイルシステムで、次の内容の `input.json` という名前のファイルを作成します。

```
{
  "motorid": "Fulton-A32",
  "sensorData": {
    "pressure": 23,
    "temperature": 47
  }
}
```

2. スターター `input.json` ファイルが作成されると、入力を作成できます。入力を作成する方法は 2 つあります。AWS IoT Events コンソールのナビゲーションペインを使用して入力を作成できます。または、ディテクターモデルの作成後に入力を作成することもできます。

入力の作成と設定

アラームモデルまたはディテクターモデルの入力を作成する方法について説明します。

1. AWS IoT Events コンソールにログインするか、新しい AWS IoT Events アカウントを作成するオプションを選択します。
2. AWS IoT Events コンソールの左上隅で、ナビゲーションペインを選択して展開します。
3. 左のナビゲーションペインで、[入力] を選択します。
4. コンソールの右隅で、[入力の作成] を選択します。
5. uniqueInputName を指定します。
6. オプション – 入力の説明を入力します。

7. JSON ファイルをアップロードするには、 の概要で作成したinput.jsonファイルを選択しますJSON 入力ファイルを作成する。入力属性を選択すると、入力した属性のリストが表示されます。
8. [入力属性の選択] では、使用する属性を選択し、[作成] を選択します。この例では、[motorid] と [sensorData.pressure] を選択しています。
9. オプション – 関連するタグを入力に追加します。

> **Note**
>
> AWS IoT Events コンソールのディテクターモデル内に追加の入力を作成することもできます。詳細については、「でディテクターモデル内に入力を作成する AWS IoT Events」を参照してください。

でディテクターモデル内に入力を作成する AWS IoT Events

のディテクター入力は、データソースとディテクターモデル間のブリッジ AWS IoT Events として機能します。ディテクター入力は、イベントの検知と自動化機能を強化する raw データを提供します AWS IoT Events。IoT エコシステムの実際のイベントや条件にモデルが正確に対応できるように、ディテクター入力を設定する方法について説明します。

このセクションでは、テレメトリデータまたはメッセージを受信するためのディテクターモデルの入力を定義する方法を示します。

ディテクターモデルの入力を定義するには

1. AWS IoT Events コンソールを開きます。
2. AWS IoT Events コンソールで、ディテクターモデルの作成を選択します。
3. [新規作成] を選択します。
4. [入力の作成] を選択します。
5. 入力には、[入力名]、オプションで [説明] を入力し、[ファイルのアップロード] を選択します。表示されるダイアログボックスで、 の概要で作成したinput.jsonファイルを選択しますJSON 入力ファイルを作成する。
6. [入力属性の選択] では、使用する属性を選択し、[作成] を選択します。この例では、motorId と sensorData.pressure を選択します。

でディテクターモデルを作成する AWS IoT Events

このトピックでは、状態を使用してディテクターモデル (機器またはプロセスのモデル) を定義します。

状態ごとに、重要なイベントを検出するために着信入力を評価する条件付き (ブール) ロジックを定義します。イベントが検出されると、状態が変化し、追加のアクションを開始できます。これらのイベントは、移行イベントと呼ばれます。

この状態では、ディテクターがその状態になるか、その状態を終了するたびに、あるいは入力を受信したときにアクションを実行できるイベント (それぞれ `OnEnter`、`OnExit`、`OnInput` イベントと呼ばれる) の定義もします 。アクションは、イベントの条件付きロジックが `true` と評価された場合にのみ実行されます。

ディテクターモデルを作成するには

1. 最初のディテクターの状態が作成されました。変更するには、メインの編集スペースで State_1 というラベルの付いた円を選択します。
2. 状態ペインで、状態名を入力し、OnEnterでイベントの追加を選択します。
3. OnEnterイベントの追加ページで、イベント名とイベント条件を入力します。この例では、`true` と入力して、状態に入ったときにイベントが常に開始されることを示します。
4. イベントアクションで、アクションの追加を選択します。
5. イベントアクションで、次の手順を実行します。

   a. 可変の設定を選択します
   b. 可変オペレーションの場合は、値の割当を選択します。
   c. 可変名に、設定する可変の名前を入力します。
   d. 可変値に、値 **`0`** (ゼロ) を入力します。
6. [Save] を選択します。

   定義した可変のような可変は、ディテクターモデルの任意のイベントで設定　(値を指定) できます。可変の値を参照できるのは、ディテクターが状態に到達し、定義または設定されているところでアクションを実行した後のみです (例えば、イベントの条件付きロジックで)。
7. 状態ペインで、状態の横のXを選択して、ディテクターモデルパレットに戻ります。
8. 2 番目のディテクターの状態を作成するには、ディテクターのモデルパレットで状態を選択し、メインの編集スペースにドラッグします。これにより、`untitled_state_1` というタイトルの状態が作成されます。

9. 最初の状態で一時停止します (通常)。状態の周囲に矢印が表示されます。
10. 矢印をクリックして、最初の状態から 2 番目の状態にドラッグします。最初の状態から 2 番目の状態 (無題のラベルが付いている) への有向線が表示されます。
11. 無題の行を選択します。移行イベントペインで、イベント名とイベントトリガーロジックを入力します。
12. 移行イベントペインで、アクションの追加を選択します。
13. 移行イベントアクションの追加ペインで、アクションの追加を選択します。
14. アクションの選択で、可変の設定を選択します。
    a. 可変オペレーションの場合は、値の割当を選択します。
    b. 可変名に、可変の名前を入力します。
    c. 値の割り当てに、次のような値を入力します: `$variable.pressureThresholdBreached + 3`
    d. [Save] を選択します。
15. 2 番目の状態 untitled_state_1 を選択します。
16. 状態ペインで状態名を入力し、入力時でイベントの追加を選択します。
17. OnEnter イベントの追加ページで、イベント名とイベント条件を入力します。[アクションを追加] を選択します。
18. アクションの選択で、SNS メッセージの送信を選択します。
    a. SNS トピックには、自分の Amazon SNS トピックのターゲット ARN を入力します。
    b. [Save] を選択します。
19. 例にイベントを追加し続けます。
    a. OnInputで、イベントの追加を選択し、次のイベント情報を入力して保存します。

```
Event name: Overpressurized
Event condition: $input.PressureInput.sensorData.pressure > 70
Event actions:
  Set variable:
    Variable operation: Assign value
    Variable name: pressureThresholdBreached
    Assign value: 3
```

    b. OnInputで、イベントの追加を選択し、次のイベント情報を入力して保存します。

```
Event name: Pressure Okay
Event condition: $input.PressureInput.sensorData.pressure <= 70
Event actions:
  Set variable:
    Variable operation: Decrement
    Variable name: pressureThresholdBreached
```

c. OnExitの場合は、イベントの追加を選択し、作成したAmazon SNS トピックの ARN を使用して、次のイベント情報を入力して保存します。

```
Event name: Normal Pressure Restored
Event condition: true
Event actions:
  Send SNS message:
    Target arn: arn:aws:sns:us-east-1:123456789012:pressureClearedAction
```

20. 2 番目の状態 (危険) で一時停止します。状態の周囲に矢印が表示されます
21. 矢印をクリックして、2 番目の状態から最初の状態にドラッグします。無題のラベルが付いた有向線が表示されます。
22. 無題の行を選択し、移行イベントペインで、次の情報を使用してイベント名とイベントトリガーロジックを入力します。

```
{
  Event name: BackToNormal
  Event trigger logic: $input.PressureInput.sensorData.pressure <= 70 &&
 $variable.pressureThresholdBreached <= 0
}
```

トリガーロジックで `$input` 値と `$variable` 値をテストする理由の詳細については、AWS IoT Events ディテクターモデルの制約と制限 での可変値の可用性のエントリを参照してください。

23. スタート状態を選択します。デフォルトでは、この状態はディテクターモデルを作成したときに作成されました)。スタートウィンドウで、宛先の状態 (例えば、通常) を選択します。
24. 次に、入力をリッスンするようにディテクターモデルを設定します。右上隅で、公開を選択します。
25. ディテクターモデルの公開ページで、次の手順を実行します。

a. ディテクターモデル名、説明、およびロールの名前を入力します。そのロールが作成されます。

b. 一意のキーバリューごとにディテクターの作成を選択します。独自のロールを作成して使用するには、のアクセス許可の設定 AWS IoT Events のステップに従って、ここにロールとして入力します。

26. ディテクター作成キーには、前に定義した入力の属性の 1 つの名前を選択します。ディテクター作成キーとして選択する属性は、各メッセージ入力に存在する必要があり、メッセージを送信する各デバイスに固有である必要があります。この例では、motorid 属性を使用しています。

27. [保存して発行] を選択します。

> **Note**
>
> 特定のディテクターモデルに対して作成される一意のディテクターの数は、送信される入力メッセージに基づいて決まります。ディテクターモデルが作成されると、入力属性からキーが選択されます。このキーは、どのディテクターインスタンスを使用するかを決定します。キーが (このディテクターモデルで) 以前に参照されたことがない場合、新しいディテクターインスタンスが作成されます。キーが以前に参照されたことがある場合は、このキー値に対応する既存のディテクターインスタンスを使用します。

ディテクターモデル定義のバックアップコピーを (JSON) 作成または更新してディテクターモデルを作成したり、テンプレートとして使用して別のディテクターモデルを作成したりできます。

これは、コンソールから、または次の CLI コマンドを使用して実行できます。必要に応じて、ディテクターモデルの名前を、前のステップで公開したときに使用したものと一致するように変更します。

```
aws iotevents describe-detector-model  --detector-model-name motorDetectorModel >
 motorDetectorModel.json
```

これにより、次のような内容のファイル (`motorDetectorModel.json`) が作成されます。

```
{
    "detectorModel": {
        "detectorModelConfiguration": {
            "status": "ACTIVE",
            "lastUpdateTime": 1552072424.212,
```

```
            "roleArn": "arn:aws:iam::123456789012:role/IoTEventsRole",
            "creationTime": 1552072424.212,
            "detectorModelArn": "arn:aws:iotevents:us-
west-2:123456789012:detectorModel/motorDetectorModel",
            "key": "motorid",
            "detectorModelName": "motorDetectorModel",
            "detectorModelVersion": "1"
        },
        "detectorModelDefinition": {
            "states": [
                {
                    "onInput": {
                        "transitionEvents": [
                            {
                                "eventName": "Overpressurized",
                                "actions": [
                                    {
                                        "setVariable": {
                                            "variableName":
 "pressureThresholdBreached",
                                            "value":
 "$variable.pressureThresholdBreached + 3"
                                        }
                                    }
                                ],
                                "condition": "$input.PressureInput.sensorData.pressure
 > 70",
                                "nextState": "Dangerous"
                            }
                        ],
                        "events": []
                    },
                    "stateName": "Normal",
                    "onEnter": {
                        "events": [
                            {
                                "eventName": "init",
                                "actions": [
                                    {
                                        "setVariable": {
                                            "variableName":
 "pressureThresholdBreached",
                                            "value": "0"
                                        }
```

```
                                    }
                                ],
                                "condition": "true"
                            }
                        ]
                    },
                    "onExit": {
                        "events": []
                    }
                },
                {
                    "onInput": {
                        "transitionEvents": [
                            {
                                "eventName": "Back to Normal",
                                "actions": [],
                                "condition": "$variable.pressureThresholdBreached <= 1
&& $input.PressureInput.sensorData.pressure <= 70",
                                "nextState": "Normal"
                            }
                        ],
                        "events": [
                            {
                                "eventName": "Overpressurized",
                                "actions": [
                                    {
                                        "setVariable": {
                                            "variableName":
"pressureThresholdBreached",
                                            "value": "3"
                                        }
                                    }
                                ],
                                "condition": "$input.PressureInput.sensorData.pressure
> 70"
                            },
                            {
                                "eventName": "Pressure Okay",
                                "actions": [
                                    {
                                        "setVariable": {
                                            "variableName":
"pressureThresholdBreached",
```

```
                                                "value":
 "$variable.pressureThresholdBreached - 1"
                                            }
                                        }
                                    ],
                                    "condition": "$input.PressureInput.sensorData.pressure
 <= 70"
                                }
                            ]
                        },
                        "stateName": "Dangerous",
                        "onEnter": {
                            "events": [
                                {
                                    "eventName": "Pressure Threshold Breached",
                                    "actions": [
                                        {
                                            "sns": {
                                                "targetArn": "arn:aws:sns:us-
west-2:123456789012:MyIoTButtonSNSTopic"
                                            }
                                        }
                                    ],
                                    "condition": "$variable.pressureThresholdBreached > 1"
                                }
                            ]
                        },
                        "onExit": {
                            "events": [
                                {
                                    "eventName": "Normal Pressure Restored",
                                    "actions": [
                                        {
                                            "sns": {
                                                "targetArn": "arn:aws:sns:us-
west-2:123456789012:IoTVirtualButtonTopic"
                                            }
                                        }
                                    ],
                                    "condition": "true"
                                }
                            ]
                        }
                    }
```

```
            ],
            "initialStateName": "Normal"
        }
    }
}
```

でディテクターモデルをテストするための入力を送信する AWS IoT Events

でテレメトリデータを受信する方法はいくつかあります AWS IoT Events（「」を参照でデータを受信してアクションをトリガーするためにサポートされているアクション AWS IoT Events）。このトピックでは、メッセージをディ AWS IoT Events テクターへの入力として転送する AWS IoT ルールを AWS IoT コンソールで作成する方法について説明します。 AWS IoT コンソールの MQTT クライアントを使用してテストメッセージを送信できます。この方法を使用すると、デバイスが AWS IoT メッセージブローカーを使用して MQTT メッセージを送信できる AWS IoT Events ときに、テレメトリデータを に取得できます。

ディテクターモデルをテストするための入力を送信するには

1. AWS IoT Core コンソールを開きます。左側のナビゲーションペインの [管理] で [メッセージルーティング] を選択し、[ルール] を選択します。
2. 右上の [ルールを作成] を選択します。
3. [ルールを作成] ページで、次の手順を完了します。

    1. ステップ 1 ルールプロパティを指定します。以下のフィールドに値を入力します。
        - ルール名。`MyIoTEventsRule` などのルールの名前を入力します。

            > **Note**
            > 空白は使用しないでください。

        - ルールの説明 これはオプションです。
        - [Next (次へ)] を選択します。
    2. ステップ 2 SQL ステートメントを設定します。以下のフィールドに値を入力します。
        - SQL のバージョン リストから適切なオプションを選択します。
        - SQL ステートメント。**`SELECT *, topic(2) as motorid FROM 'motors/+/status'`** と入力してください。

[次へ] を選択します。

3. ステップ 3 ルールアクションをアタッチします。「ルールアクション」セクションで、以下を完了します。

   - アクション 1 IoT イベントを選択します。次のフィールドが表示されます。

     a. 入力名 リストから適切なオプションを選択します。自分の入力が表示されない場合は、[更新] を選択します。

     新しい入力を作成するには、[IoT イベント入力を作成] を選択します。以下のフィールドに値を入力します。

     - 入力名 `PressureInput` と入力してください。
     - 説明。これはオプションです。
     - JSON ファイルをアップロードします。JSON ファイルのコピーをアップロードします。ファイルがない場合は、この画面上のサンプルファイルへのリンクが表示されます。コードには以下が含まれます。

     ```
     {
       "motorid": "Fulton-A32",
       "sensorData": {
         "pressure": 23,
         "temperature": 47
       }
     }
     ```

     - 入力属性を選択します。適切なオプションを選択します。
     - タグ これはオプションです。

     [作成] を選択します。

     [ルールを作成] 画面に戻り、[入力名] フィールドを更新します。　作成したばかりの入力を選択します。

     b. バッチモード これはオプションです。ペイロードがメッセージの配列の場合は、このオプションを選択します。

     c. メッセージ ID これはオプションですが推奨されます。

     d. IAM ロール リストから適切なロールを選択します。ロールがリストにない場合は、[新しいロールを作成] を選択します。

     ロール名を入力し、[作成] を選択します。

別のルールを追加するには、[ルールアクションを追加] を選択します。

- エラーアクション このセクションはオプションです。アクションを追加するには、[エラーアクションを追加] を選択し、リストから適切なアクションを選択します。

  表示されるフィールドに入力します。

- [Next (次へ)] を選択します。

4. ステップ 4 確認して作成します。画面上の情報を確認し、[作成] を選択します。

4. 左側のナビゲーションメニューの [テスト] で、[MQTT テストクライアント] を選択します。
5. [トピックに発行] を選択します。以下のフィールドに値を入力します。

- トピック名 `motors/Fulton-A32/status` などのメッセージを識別する名前を入力します。
- メッセージのペイロード。次のように入力します。

```
{
  "messageId": 100,
  "sensorData": {
    "pressure": 39
  }
}
```

> **Note**
>
> 新しいメッセージを発行するたびに、`messageId` を変更します。

6. 発行の場合、トピックは同じままにしますが、ペイロードの `"pressure"` を、ディテクターモデルで指定したしきい値 (**85**など) よりも大きい値に変更します。
7. [発行] を選択します。

作成したディテクターインスタンスは、Amazon SNS メッセージを生成して送信します。圧力しきい値 (この例では 70) を超えるまたは下回る圧力測定値を含むメッセージを送信し続けて、ディテクターがオペレーションしていることを確認します。

この例では、圧力測定値がしきい値を下回る 3 つのメッセージを送信して、正常状態に戻り、過圧状態が解消されたことを示す Amazon SNS メッセージを受信する必要があります。正常状態に戻ると、圧力測定値が制限を超える 1 つのメッセージにより、ディテクターが危険な状態になり、その状態を示す Amazon SNS メッセージが送信されます。

簡単な入力とディテクターモデルを作成したので、次のことを試してください。

- コンソールで他のディテクターモデルの例 (テンプレート) を参照してください。
- 「」のステップに従ってCLI を使用して 2 つの状態の AWS IoT Events ディテクターを作成する、を使用して入力モデルとディテクターモデルを作成します。 AWS CLI
- イベントで使用される イベントデータをフィルタリング、変換、処理する式 の詳細を学びます。
- でデータを受信してアクションをトリガーするためにサポートされているアクション AWS IoT Events　についてはこちら。
- 何かが機能していない場合は、トラブルシューティング AWS IoT Events を参照してください。

のベストプラクティス AWS IoT Events

AWS IoT Eventsから最大の利点を得るには、これらのベストプラクティスに従ってください。

トピック

- AWS IoT Events ディテクターモデルの開発時に Amazon CloudWatch ログ記録を有効にする
- AWS IoT Events コンソールで作業するときにディテクターモデルを保存するために定期的に公開する

AWS IoT Events ディテクターモデルの開発時に Amazon CloudWatch ログ記録を有効にする

Amazon CloudWatch は、 AWS リソースと AWS で実行しているアプリケーションをリアルタイムでモニタリングします。CloudWatch を使用すると、リソースの使用、アプリケーションのパフォーマンス、運用のヘルスについてシステム全体の可視性を得ることができます。 AWS IoT Events ディテクターモデルを開発またはデバッグすると、CloudWatch AWS IoT Events は何が行われているか、および発生したエラーを把握するのに役立ちます。

CloudWatch を有効にするには

1. まだ作成していない場合は、「」の手順に従ってのアクセス許可の設定 AWS IoT Events、CloudWatch ログを作成および管理するためのアクセス許可を付与するポリシーがアタッチされたロールを作成します AWS IoT Events。
2. [AWS IoT Events コンソール] に移動します。
3. ナビゲーションペインで [Settings] (設定) を選択します。
4. [Settings] (設定) ページで、[Edit] (編集) を選択します。
5. ログ記録オプションの編集ページのログ記録オプションセクションで、次の操作を行います。
   a. 詳細レベル で、オプションを選択します。
   b. [Select role] (ロールを選択) で、選択したログ記録アクションを実行するのに十分な許可を持つロールを選択します。
   c. (オプション) 詳細レベルにデバッグを選択した場合は、次の手順を実行してデバッグターゲットを追加できます。
      i. デバッグターゲットで、モデルオプションの追加を選択します。

ii. ディテクターモデル名と (オプション) KeyValue を入力して、ログに記録するディテクターモデルと特定のディテクター (インスタンス) を指定します。

6. [Update] (更新) を選択します。

ログオプションが正常に更新されました。

AWS IoT Events コンソールで作業するときにディテクターモデルを保存するために定期的に公開する

AWS IoT Events コンソールを使用すると、進行中の作業はブラウザにローカルに保存されます。ただし、ディテクータモデルを AWS IoT Eventsに保存するには、[Publish] (発行)を選択する必要があります。ディテクターモデルを発行すると、発行された作業結果は、自分のアカウントへアクセスしているどのブラウザでも利用できるようになります。

> **Note**
>
> 作業結果を発行しないと、保存はされません。ディテクターモデルを一旦発行すると、名前を変更することはできません。ただし、その定義を変更し続けることができます。

AWS IoT Events ユースケースのチュートリアル

AWS IoT Events チュートリアルでは、基本的なセットアップからより具体的なユースケースまで AWS IoT Events、 のさまざまな側面をカバーする一連の手順を提供します。各チュートリアルでは、実際のシナリオの例を示し、ディテクターモデルの作成、入力の設定、アクションの設定、および強力な IoT ソリューションを作成するための他の AWS サービスとの統合に関する実際のスキルを構築するのに役立ちます。

この章では、次の方法について説明します。

- ディテクターモデルに含める状態を決定し、1 つのディテクターインスタンスが必要か複数のディテクターインスタンスが必要かを判断するためのヘルプを入手してください。
- AWS CLIを使用する例に従ってください。
- デバイスからテレメトリデータを受信するための入力と、そのデータを送信するデバイスの状態をモニタリングおよびレポートするためのディテクターモデルを作成します。
- 入力、ディテクターモデル、および AWS IoT Events サービスに関する制限と制限を確認します。
- コメントを含めた、ディテクターモデルのより複雑な例を参照してください。

トピック

- AWS IoT Events を使用して IoT デバイスを監視する
- CLI を使用して 2 つの状態の AWS IoT Events ディテクターを作成する
- AWS IoT Events ディテクターモデルの制約と制限
- コメントされた例: を使用した HIPAA 温度制御 AWS IoT Events

AWS IoT Events を使用して IoT デバイスを監視する

AWS IoT Events を使用してデバイスまたはプロセスをモニタリングし、重要なイベントに基づいてアクションを実行できます。これを行うには、次のベーシックなステップに従います。

入力を作成する

デバイスとプロセスがテレメトリデータを AWS IoT Eventsに取り込む方法が必要です。これを行うには、 AWS IoT Eventsへの入力としてメッセージを送信します。いくつかの方法で入力としてメッセージを送信できます。

- BatchPutMessageオペレーションを使用します。

- AWS IoT Core ルールエンジンの `iotEvents` ルールアクションを定義します。ルールアクションは、入力から AWS IoT Eventsにメッセージデータを転送します。
- で AWS IoT Analytics、CreateDataset オペレーションを使用して、 でデータセットを作成しますcontentDeliveryRules。これらのルールは、データセットコンテンツが自動的に送信される AWS IoT Events 入力を指定します。
- AWS IoT Events ディテクターモデルの onInput、、onExitまたは transitionEventsイベントで iotEvents アクションを定義します。ディテクターモデルインスタンスとアクションを開始したイベントに関する情報は、指定した名前の入力としてシステムにフィードバックされます。

デバイスがこの方法でデータの送信をスタートする前に、1 つ以上の入力を定義する必要があります。これを行うには、各入力に名前を付け、入力が監視する受信メッセージデータのフィールドを指定します。 は、JSON ペイロード形式で、多くのソースから入力 AWS IoT Events を受け取ります。各入力は、単独で操作することも、他の入力と組み合わせて、より複雑なイベントを検出することもできます。

ディテクターモデルを作成する

状態を使用して、ディテクターモデル (機器またはプロセスのモデル) を定義します。状態ごとに、重要なイベントを検出するために着信入力を評価する条件付き (ブール) ロジックを定義します。イベントが検出されると、状態を変更したり、他の AWS サービスを使用してカスタムビルドまたは事前定義されたアクションを開始したりできます。状態に入るときまたは状態を出るとき、およびオプションで条件が満たされたときにアクションを開始する追加のイベントを定義できます。

このチュートリアルでは、モデルが特定の状態に出入りするときのアクションとして Amazon SNS メッセージを送信します。

デバイスまたはプロセスをモニタリングする

複数のデバイスまたはプロセスをモニタリングしている場合は、入力元の特定のデバイスまたはプロセスを識別するフィールドを各入力に指定します。(`CreateDetectorModel` の `key` フィールドを参照してください。) 新しいデバイスが識別されると (`key` によって識別される入力フィールドに新しい値が表示されます)、ディテクターが作成されます。(各ディテクターは、ディテクターモデルのインスタンスです。) 次に、新しいディテクターは、ディテクターモデルが更新または削除されるまで、そのデバイスからの入力に応答し続けます。

単一のプロセスをモニタリングしている場合 (複数のデバイスまたはサブプロセスが入力を送信している場合でも)、一意の識別 `key` フィールドを指定しません。この場合、最初の入力が到着したときに単一のディテクター (インスタンス) が作成されます。

ディテクターモデルへの入力としてメッセージを送信します

デバイスまたはプロセスからディ AWS IoT Events テクターへの入力としてメッセージを送信するには、メッセージに追加のフォーマットを実行する必要がない方法がいくつかあります。このチュートリアルでは、 AWS IoT コンソールを使用して、メッセージデータを転送するルールエンジンのAWS IoT Events アクション AWS IoT Core ルールを記述します AWS IoT Events。これを行うには、入力を名前で識別します。次に、コンソールを使用して AWS IoT 、入力として転送されるメッセージをいくつか生成します AWS IoT Events。

ディテクターモデルに必要な状態をどのようにして知ることができますか？

ディテクターモデルが持つべき状態を決定するには、最初に実行できるアクションを決定します。例えば、自動車がガソリンで走行している場合、旅行をスタートするときに燃料計を見て、燃料を補給する必要があるかどうかを確認します。ここに 1 つのアクションがあります:「ガスを取りに行く」ようにドライバーに伝えます。ディテクターモデルには、「車は燃料を必要としない」と「車は燃料を必要とする」の 2 つの状態が必要です。一般に、可能なアクションごとに 1 つの状態を定義し、さらにアクションが不要な場合にもう 1 つの状態を定義する必要があります。これは、アクション自体がより複雑な場合でも機能します。例えば、最寄りのガソリンステーションの場所や最も安い料金に関する情報を調べて含めることができますが、これは「ガソリンを取りに行く」というメッセージを送信するときに行います。

次に入る状態を決定するには、入力を確認します。入力には、どの状態にするかを決定するために必要な情報が含まれています。入力を作成するには、デバイスまたはプロセスによって送信されるメッセージ内の 1 つ以上のフィールドを選択して、決定に役立てます。この例では、現在の燃料レベル (「フルパーセント」) を示す 1 つの入力が必要です。おそらくお客様の車はいくつかの異なるメッセージを送っていて、それぞれがいくつかの異なるフィールドを持っています。この入力を作成するには、現在のガスゲージレベルを報告するメッセージとフィールドを選択する必要があります。移動しようとしている旅行の長さ (「目的地までの距離」) は、物事を単純にするためにハードコーディングすることができます。平均的な旅行の長さを使うことができます。入力に基づいていくつかの計算を行います (そのパーセントが完全に変換されるのは何ガロンですか？これは、ガロンと平均「ガロンあたりのマイル」を考慮して、移動できるマイルよりも長い平均旅行距離です)。これらの計算を実行し、イベントでメッセージを送信します。

これまでのところ、2 つの状態と 1 つの入力があります。入力に基づいて計算を実行し、2 番目の状態に進むかどうかを決定する最初の状態のイベントが必要です。それは移行イベントです。(`transitionEvents` は状態の `onInput` イベントリストにあります。この最初の状態で入力を受

信すると、イベントの condition が満たされると、イベントは 2 番目の状態への移行を実行します。) 2 番目の状態に達すると、状態に入るとすぐにメッセージを送信します。(onEnter イベントを使用します。2 番目の状態に入ると、このイベントはメッセージを送信します。別の入力が到着するのを待つ必要はありません。) 他の種類のイベントもありますが、簡単な例として必要なのはそれだけです。

他のタイプのイベントは onExit と onInput です。入力を受け取り、条件が満たされるとすぐに、onInput イベントは指定されたアクションを実行します。オペレーションが現在の状態を終了し、条件が満たされると、onExit イベントは、指定されたアクションを実行します。

何か足りないものはありますか？ はい、どうすれば最初の「車は燃料を必要としない」状態に戻ることができますか？ ガスタンクを満たした後、入力は満タンを示します。2 番目の状態では、入力が受信されたときに発生する最初の状態に戻る移行イベントが必要です (2 番目の状態の onInput: イベント)。計算により、目的の場所に到達するのに十分なガスがあることが示された場合は、最初の状態に戻るはずです。

それが基本です。一部のディテクターモデルは、可能なアクションだけでなく、重要な入力を反映する状態を追加することにより、より複雑になります。例えば、温度を追跡するディテクターモデルに、「正常」状態、「高温」状態、「潜在的な問題」状態の 3 つの状態があるとします。温度が特定のレベルを超えると、潜在的な問題の状態に移行しますが、まだ熱くなりすぎていません。この温度に 15 分以上留まらない限り、アラームを送信したくありません。それ以前に温度が正常に戻ると、ディテクターは正常状態に戻ります。タイマーが時間切れになると、ディテクターは高すぎる状態に移行し、アラームを送信しますが、注意が必要です。可変とより複雑なイベント条件のセットを使用して、同じことを行うことができます。ただし、多くの場合、別の状態を使用して、実際には計算結果を保存する方が簡単です。

ディテクターのインスタンスが 1 つ必要か、それとも複数必要かをどのようにして知ることができますか？

必要なインスタンスの数を決定するには、「何を知りたいですか？」と自問してください。今日の天気を知りたいとしましょう。雨が降っていますか (状態)？ 傘をさす必要がありますか (アクション)？ 温度を報告するセンサー、湿度を報告するセンサー、気圧、風速と風向、降水量を報告するセンサーを使用できます。ただし、これらすべてのセンサーを一緒にモニタリングして、天気の状態 (雨、雪、曇り、晴れ) と適切なアクション (傘をさすか日焼け止めを塗る) を判断する必要があります。センサーの数にかかわらず、1 つのディテクターインスタンスで天気の状態をモニタリングし、実行するアクションを通知する必要があります。

ただし、お住まいのリージョンの天気予測をしている場合は、そのようなセンサーアレイの複数のインスタンスが、リージョン全体のさまざまな場所に配置されている可能性があります。各場所の人々

は、その場所の天気がどのようなものかを知る必要があります。この場合、ディテクターの複数のインスタンスが必要です。各場所の各センサーによって報告されるデータには、key フィールドとして指定したフィールドが含まれている必要があります。このフィールドにより、 AWS IoT Events はエリアのディテクターインスタンスを作成し、到着し続けるときにこの情報をそのディテクターインスタンスに送信を続けることができます。髪を荒らしたり、鼻を日焼けさせたりする必要はもうありません！

基本的に、モニタリングする状況が 1 つ (1 つのプロセスまたは 1 つの場所) ある場合は、1 つのディテクターインスタンスが必要です。モニタリングする状況 (場所、プロセス) が多い場合は、複数のディテクターインスタンスが必要です。

CLI を使用して 2 つの状態の AWS IoT Events ディテクターを作成する

この例では、 AWS CLI コマンドを使用して AWS IoT Events APIs を呼び出し、エンジンの 2 つの状態をモデル化するディテクターを作成します。1 つは通常の状態、もう 1 つは過圧状態です。

エンジン内の測定された圧力が特定のしきい値を超えると、モデルは過圧状態に移行し、Amazon Simple Notification Service (Amazon SNS) メッセージを送信して、技術者にその状態を警告します。圧力が 3 回連続して圧力測定値のしきい値を下回ると、モデルは通常の状態に戻り、状態がクリアされたことを確認するために別の Amazon SNS メッセージを送信します。非線形回復フェーズまたは 1 回限りの異常な回復読み取りの場合に、過圧/正常メッセージの吃音の可能性を排除するために、圧力しきい値を下回る 3 つの連続した読み取りが必要です。

以下は、ディテクターを作成するステップの概要です。

入力の作成。

デバイスとプロセスをモニタリングするには、テレメトリデータを AWS IoT Eventsに取り込む方法が必要です。これを行うには、メッセージを入力として に送信します AWS IoT Events。これはいくつかの方法で行うことができます。

- BatchPutMessageオペレーションを使用します。この方法は簡単ですが、デバイスまたはプロセスが SDK または を介して AWS IoT Events API にアクセスできる必要があります AWS CLI。
- で AWS IoT Core、メッセージデータを転送するルールエンジンのAWS IoT Events アクション AWS IoT Core ルールを記述します AWS IoT Events。これは、入力を名前で識別します。デバイスまたはプロセスが を介してメッセージを送信できる、または既に送信している場合は、こ

の方法を使用します AWS IoT Core。この方法では、通常、デバイスからの処理能力が少なくて済みます。

- では AWS IoT Analytics、CreateDataset オペレーションを使用して、データセットの内容が自動的に送信される AWS IoT Events 入力`contentDeliveryRules`を指定するデータセットをで作成します。 AWS IoT Analyticsで集約または分析されたデータに基づいてデバイスまたはプロセスを制御する場合は、この方法を使用します。

デバイスがこの方法でデータを送信する前に、1 つ以上の入力を定義する必要があります。これを行うには、各入力に名前を付け、入力がモニタリングする着信メッセージデータのフィールドを指定します。

ディテクターモデルを作成する

状態を使用してディテクターモデル (機器またはプロセスのモデル) を作成します。状態ごとに、重要なイベントを検出するために着信入力を評価する条件付き (ブール) ロジックを定義します。イベントが検出されると、状態を変更したり、他の AWS サービスを使用してカスタムビルドまたは事前定義されたアクションを開始したりできます。状態に入るときまたは状態を出るとき、およびオプションで条件が満たされたときにアクションを開始する追加のイベントを定義できます。

複数のデバイスまたはプロセスをモニタリングする

複数のデバイスまたはプロセスをモニタリングしていて、それぞれを個別に追跡する場合は、入力元の特定のデバイスまたはプロセスを識別するフィールドを各入力に指定します。`CreateDetectorModel` の `key` フィールドを参照してください。新しいデバイスが識別されると (`key`によって識別される入力フィールドに新しい値が表示されます)、ディテクターインスタンスが作成されます。新しいディテクターインスタンスは、ディテクターモデルが更新または削除されるまで、その特定のデバイスからの入力に応答し続けます。入力 `key` フィールドに一意の値があるのと同じ数の一意のディテクター (インスタンス) があります。

単一のデバイスまたはプロセスをモニタリングする

単一のプロセスをモニタリングしている場合 (複数のデバイスまたはサブプロセスが入力を送信している場合でも)、一意の識別 `key` フィールドを指定しません。この場合、最初の入力が到着したときに単一のディテクター (インスタンス) が作成されます。例えば、家の各部屋に温度センサーがあり、家全体を暖房または冷房するための HVAC 単位は 1 つだけである場合があります。したがって、各部屋の居住者が自分の投票 (入力) を優先することを望んでいる場合でも、これを単一のプロセスとしてのみ制御できます。

デバイスまたはプロセスからのメッセージをディテクターモデルへの入力として送信します

デバイスまたはプロセスから、入力内の AWS IoT Events ディテクターへの入力としてメッセージを送信するいくつかの方法について説明しました。入力を作成してディテクターモデルを構築したら、データの送信をスタートする準備が整います。

> **Note**
>
> ディテクターモデルを作成する場合、または既存のモデルを更新する場合、新しいディテクターモデルまたは更新されたディテクターモデルがメッセージの受信とディテクター (インスタンス) の作成を開始するまでに数分かかります。ディテクターモデルが更新された場合、この間、以前のバージョンに基づく動作が引き続き表示される可能性があります。

トピック

- デバイスデータをキャプチャする AWS IoT Events 入力を作成する
- でデバイスの状態を表すディテクターモデルを作成する AWS IoT Events
- でディテクターへの入力としてメッセージを送信する AWS IoT Events

デバイスデータをキャプチャする AWS IoT Events 入力を作成する

の入力を設定するときは AWS IoT Events、 を活用して AWS CLI 、デバイスがセンサーデータをどのように通信するかを定義できます。例えば、デバイスがモーター識別子とセンサーの読み取り値を含む JSON 形式のメッセージを送信する場合、圧力やモーター ID など、メッセージからの特定の属性をマッピングする入力を作成することで、このデータをキャプチャできます。このプロセスは、JSON ファイルで入力を定義し、関連するデータポイントを指定し、 を使用して入力を登録 AWS CLI することから始まります AWS IoT Events。これにより AWS IoT 、 はリアルタイムのセンサーデータに基づいて、重要な状態をモニタリングして対応できます。

例として、デバイスが次の形式でメッセージを送信するとします。

```
{
  "motorid": "Fulton-A32",
  "sensorData": {
    "pressure": 23,
    "temperature": 47
  }
```

```
}
```

次の AWS CLI コマンドを使用して、pressureデータと motorid (メッセージを送信した特定のデバイスを識別する) をキャプチャする入力を作成できます。

```
aws iotevents create-input  --cli-input-json file://pressureInput.json
```

ファイル pressureInput.json には次のものが含まれています。

```
{
  "inputName": "PressureInput",
  "inputDescription": "Pressure readings from a motor",
  "inputDefinition": {
    "attributes": [
      { "jsonPath": "sensorData.pressure" },
      { "jsonPath": "motorid" }
    ]
  }
}
```

独自の入力を作成するときは、最初にサンプルメッセージをデバイスまたはプロセスから JSON ファイルとして収集することを忘れないでください。それらを使用して、コンソールまたは CLI から入力を作成できます。

でデバイスの状態を表すディテクターモデルを作成する AWS IoT Events

デバイスデータをキャプチャする AWS IoT Events 入力を作成する では、モーターからの圧力データを報告するメッセージに基づいて input を作成しました。例を続けるために、これはモーターの過圧イベントに応答するディテクターモデルです。

「Normal」と「Dangerous」の 2 つの状態を作成します。各ディテクター (インスタンス) は、作成時に「Normal」状態になります。インスタンスは、key「motorid」の一意の値を持つ入力が到着したときに作成されます。

ディテクターインスタンスが 70 以上の圧力測定値を受信すると、「Dangerous」状態になり、警告として Amazon SNS メッセージを送信します。3 回の連続入力で圧力測定値が通常 (70 未満) に戻ると、ディテクターは「Normal」状態に戻り、別の Amazon SNS メッセージをすべてクリアとして送信します。

このディテクターモデルの例では、Amazon リソースネーム (ARN) が `"targetArn": "arn:aws:sns:us-east-1:123456789012:underPressureAction"` および `"targetArn": "arn:aws:sns:us-east-1:123456789012:pressureClearedAction"` として定義に示されている 2 つの Amazon SNS トピックを作成したことを前提としています。

詳細については、Amazon Simple Notification Service デベロッパーガイドを参照してください。具体的には、Amazon Simple Notification Service API リファレンス の CreateTopic オペレーションのドキュメントを参照してください。

この例では、適切なアクセス許可を持つ AWS Identity and Access Management (IAM) ロールを作成していることも前提としています。このロールの ARN は、ディテクターモデルの定義に `"roleArn": "arn:aws:iam::123456789012:role/IoTEventsRole"` として示されています。のアクセス許可の設定 AWS IoT Events のステップに従ってこのロールを作成し、ロールの ARN をディテクターモデル定義の適切な場所にコピーします。

次の AWS CLI コマンドを使用してディテクターモデルを作成できます。

```
aws iotevents create-detector-model  --cli-input-json file://motorDetectorModel.json
```

ファイル `"motorDetectorModel.json"` には次のものが含まれています。

```
{
  "detectorModelName": "motorDetectorModel",
  "detectorModelDefinition": {
    "states": [
      {
        "stateName": "Normal",
        "onEnter": {
          "events": [
            {
              "eventName": "init",
              "condition": "true",
              "actions": [
                {
                  "setVariable": {
                    "variableName": "pressureThresholdBreached",
                    "value": "0"
                  }
                }
              ]
            }
```

```
          ]
        },
        "onInput": {
          "transitionEvents": [
            {
              "eventName": "Overpressurized",
              "condition": "$input.PressureInput.sensorData.pressure > 70",
              "actions": [
                {
                  "setVariable": {
                    "variableName": "pressureThresholdBreached",
                    "value": "$variable.pressureThresholdBreached + 3"
                  }
                }
              ],
              "nextState": "Dangerous"
            }
          ]
        }
      },
      {
        "stateName": "Dangerous",
        "onEnter": {
          "events": [
            {
              "eventName": "Pressure Threshold Breached",
              "condition": "$variable.pressureThresholdBreached > 1",
              "actions": [
                {
                  "sns": {
                    "targetArn": "arn:aws:sns:us-
east-1:123456789012:underPressureAction"
                  }
                }
              ]
            }
          ]
        },
        "onInput": {
          "events": [
            {
              "eventName": "Overpressurized",
              "condition": "$input.PressureInput.sensorData.pressure > 70",
              "actions": [
```

```
                {
                  "setVariable": {
                    "variableName": "pressureThresholdBreached",
                    "value": "3"
                  }
                }
              ]
            },
            {
              "eventName": "Pressure Okay",
              "condition": "$input.PressureInput.sensorData.pressure <= 70",
              "actions": [
                {
                  "setVariable": {
                    "variableName": "pressureThresholdBreached",
                    "value": "$variable.pressureThresholdBreached - 1"
                  }
                }
              ]
            }
          ],
          "transitionEvents": [
            {
              "eventName": "BackToNormal",
              "condition": "$input.PressureInput.sensorData.pressure <= 70 &&
 $variable.pressureThresholdBreached <= 1",
              "nextState": "Normal"
            }
          ]
        },
        "onExit": {
          "events": [
            {
              "eventName": "Normal Pressure Restored",
              "condition": "true",
              "actions": [
                {
                  "sns": {
                    "targetArn": "arn:aws:sns:us-
east-1:123456789012:pressureClearedAction"
                  }
                }
              ]
            }
```

```
          ]
        }
      }
    ],
    "initialStateName": "Normal"
  },
  "key" : "motorid",
  "roleArn": "arn:aws:iam::123456789012:role/IoTEventsRole"
}
```

でディテクターへの入力としてメッセージを送信する AWS IoT Events

これで、デバイスから送信されるメッセージの重要なフィールドを識別する入力が定義されました (デバイスデータをキャプチャする AWS IoT Events 入力を作成する を参照)。前のセクションでは、モーターの過圧イベントに応答する `detector model` を作成しました (でデバイスの状態を表すディテクターモデルを作成する AWS IoT Events を参照)。

例を完了するには、デバイス (この場合は AWS CLI がインストールされているコンピュータ) からのメッセージをディテクターへの入力として送信します。

> **Note**
>
> ディテクターモデルを作成するか、既存のモデルを更新する場合、新しいディテクターモデルまたは更新されたディテクターモデルがメッセージの受信とディテクター (インスタンス) の作成を開始するまでに数分かかります。ディテクターモデルを更新すると、この間、以前のバージョンに基づく動作が引き続き表示される場合があります。

次の AWS CLI コマンドを使用して、しきい値を超えるデータを含むメッセージを送信します。

```
aws iotevents-data batch-put-message --cli-input-json file://highPressureMessage.json
 --cli-binary-format raw-in-base64-out
```

ファイル「`highPressureMessage.json`」には以下が含まれています。

```
{
  "messages": [
    {
      "messageId": "00001",
```

```
      "inputName": "PressureInput",
      "payload": "{\"motorid\": \"Fulton-A32\", \"sensorData\": {\"pressure\": 80,
 \"temperature\": 39} }"
    }
  ]
}
```

送信される各メッセージの messageId を変更する必要があります。変更しない場合、 AWS IoT Events システムはメッセージを重複排除します。過去 5 分以内に送信された別のメッセージmessageIDと同じメッセージがある場合は、メッセージ AWS IoT Events を無視します。

この時点で、モーター "Fulton-A32" のイベントをモニタリングするためのディテクター (インスタンス) が作成されます。このディテクターは、作成時に "Normal" 状態になります。ただし、しきい値を超える圧力値を送信したため、すぐに "Dangerous" 状態に移行します。その際、ディテクターは、ARN が arn:aws:sns:us-east-1:123456789012:underPressureAction である Amazon SNS エンドポイントにメッセージを送信します。

次の AWS CLI コマンドを実行して、圧力しきい値を下回るデータを含むメッセージを送信します。

```
aws iotevents-data batch-put-message --cli-input-json file://normalPressureMessage.json
 --cli-binary-format raw-in-base64-out
```

ファイル normalPressureMessage.json には次のものが含まれています。

```
{
  "messages": [
    {
      "messageId": "00002",
      "inputName": "PressureInput",
      "payload": "{\"motorid\": \"Fulton-A32\", \"sensorData\": {\"pressure\": 60,
 \"temperature\": 29} }"
    }
  ]
}
```

5 分以内に BatchPutMessage コマンドを呼び出すたびに、ファイル内の messageId を変更する必要があります。メッセージをさらに 2 回送信します。メッセージが 3 回送信された後、モーター「Fulton-A32」のディテクター (インスタンス) がAmazon SNS エンドポイント "arn:aws:sns:us-east-1:123456789012:pressureClearedAction" にメッセージを送信し、"Normal" 状態に戻ります。

> **Note**
>
> `BatchPutMessage` を使用すると、一度に複数のメッセージを送信できます。ただし、これらのメッセージが処理される順序は保証されません。メッセージ (入力) が順番に処理されることを保証するには、メッセージを一度に 1 つずつ送信し、API が呼び出されるたびに正常なレスポンスを待ちます。

以下は、このセクションで説明するディテクターモデルの例によって作成された SNS メッセージペイロードの例です。

イベント「圧力しきい値違反」

```
IoT> {
  "eventTime":1558129816420,
  "payload":{
    "actionExecutionId":"5d7444df-a655-3587-a609-dbd7a0f55267",
    "detector":{
      "detectorModelName":"motorDetectorModel",
      "keyValue":"Fulton-A32",
      "detectorModelVersion":"1"
    },
    "eventTriggerDetails":{
      "inputName":"PressureInput",
      "messageId":"00001",
      "triggerType":"Message"
    },
    "state":{
      "stateName":"Dangerous",
      "variables":{
        "pressureThresholdBreached":3
      },
      "timers":{}
    }
  },
  "eventName":"Pressure Threshold Breached"
}
```

イベント「通常の圧力が回復しました」

```
IoT> {
  "eventTime":1558129925568,
```

```
  "payload":{
    "actionExecutionId":"7e25fd38-2533-303d-899f-c979792a12cb",
    "detector":{
      "detectorModelName":"motorDetectorModel",
      "keyValue":"Fulton-A32",
      "detectorModelVersion":"1"
    },
    "eventTriggerDetails":{
      "inputName":"PressureInput",
      "messageId":"00004",
      "triggerType":"Message"
    },
    "state":{
      "stateName":"Dangerous",
      "variables":{
        "pressureThresholdBreached":0
      },
      "timers":{}
    }
  },
  "eventName":"Normal Pressure Restored"
}
```

タイマーを定義した場合、それらの現在の状態は SNS メッセージペイロードにも表示されます。

メッセージペイロードには、メッセージが送信されたとき (つまり、SNS アクションが実行されたとき) のディテクター (インスタンス) の状態に関する情報が含まれています。https://docs.aws.amazon.com/iotevents/latest/apireference/API_iotevents-data_DescribeDetector.html オペレーションを使用して、ディテクターの状態に関する同様の情報を取得できます。

AWS IoT Events ディテクターモデルの制約と制限

ディテクターモデルを作成する際には、次の点を考慮することが重要です。

actions フィールドの使用方法

actions フィールドは、オブジェクトのリストです。複数のオブジェクトを持つことができますが、各オブジェクトで許可されるアクションは 1 つだけです。

Example

```
                "actions": [
                  {
                    "setVariable": {
                      "variableName": "pressureThresholdBreached",
                      "value": "$variable.pressureThresholdBreached - 1"
                    }
                  }
                  {
                    "setVariable": {
                      "variableName": "temperatureIsTooHigh",
                      "value": "$variable.temperatureIsTooHigh - 1"
                    }
                  }
                ]
```

condition フィールドの使用方法

condition は transitionEvents に必須であり、それ以外の場合はオプションです。

condition フィールドが存在しない場合は、"condition": true と同等です。

条件表現の評価結果はブール値である必要があります。結果がブール値でない場合、false と同等であり、actions を開始したり、イベントで指定された nextState に移行したりすることはありません。

可変値の可用性

デフォルトでは、可変の値がイベントに設定されている場合、その新しい値は使用できないか、同じグループ内の他のイベントの条件を評価するために使用されません。新しい値は使用できないか、同じ onInput、onEnter、または onExit フィールドのイベント条件で使用されません。

この動作を変更するには、ディテクターモデル定義で evaluationMethodパラメータを設定します。evaluationMethod が SERIAL に設定されている場合、可変が更新され、イベントが定義された順序でイベント条件が評価されます。それ以外の場合、evaluationMethod が BATCH に設定されているか、デフォルトで設定されていると、状態内の可変が更新され、状態内のイベントはすべてのイベント条件が評価された後にのみ実行されます。

"Dangerous" 状態の onInput フィールドでは、条件が満たされたとき (現在の入力の圧力が 70 以下のとき)、"Pressure Okay" イベントで "$variable.pressureThresholdBreached" が 1 つデクリメントされます。

```
            {
              "eventName": "Pressure Okay",
              "condition": "$input.PressureInput.sensorData.pressure <= 70",
              "actions": [
                {
                  "setVariable": {
                    "variableName": "pressureThresholdBreached",
                    "value": "$variable.pressureThresholdBreached - 1"
                  }
                }
              ]
            }
```

"$variable.pressureThresholdBreached" が 0 に達すると (つまり、ディテクターが 70 以下の 3 つの連続した圧力測定値を受け取った場合)、ディテクターは "Normal" 状態に戻る必要があります。transitionEvents の "BackToNormal" イベントは、"$variable.pressureThresholdBreached" が 1 以下 (0 ではない) であることをテストし、さらに "$input.PressureInput.sensorData.pressure" で与えられる現在の値が 70 以下であることを再度検証しなければなりません。

```
          "transitionEvents": [
            {
              "eventName": "BackToNormal",
              "condition": "$input.PressureInput.sensorData.pressure <= 70 &&
 $variable.pressureThresholdBreached <= 1",
              "nextState": "Normal"
            }
          ]
```

それ以外の場合、条件が可変の値のみをテストする場合、2 つの通常の読み取り値とそれに続く過圧読み取り値が条件を満たし、"Normal" 状態に戻ります。条件は、前回入力が処理されたときに "$variable.pressureThresholdBreached" が与えられた値を調べています。可変の値は "Overpressurized" イベントで 3 にリセットされますが、この新しい値はまだどの condition でも使用できないことに注意してください。

デフォルトでは、コントロールが onInput フィールドに入るたびに、condition は、onInput で指定されたアクションによって変更される前に、入力の処理のスタート時の可変の値のみを確認できます。同じことが onEnter と onExit にも当てはまります。状態に入ると

きまたは状態を終了するときに可変に加えられた変更は、同じ onEnter または onExit フィールドで指定された他の条件では使用できません。

ディテクターモデルを更新するときのレイテンシー

ディテクターモデルを更新、削除、および再作成する場合 (UpdateDetectorModel を参照)、生成されたすべてのディテクター (インスタンス) が削除され、新しいモデルがディテクターの再作成に使用されるまでに、ある程度の遅延があります。これらは、新しいディテクターモデルが有効になり、新しい入力が到着した後に再作成されます。この間、入力は、以前のバージョンのディテクターモデルによって生成されたディテクターによって引き続き処理される可能性があります。この期間中、以前のディテクターモデルで定義されたアラートを引き続き受信する可能性があります。

入力キーのスペース

入力キーにはスペースを使用できますが、入力属性の定義と、キーの値が表現で参照される場合の両方で、キーへのリファレンスをバッククォートで囲む必要があります。例えば、次のようなメッセージペイロードがあるとします。

```
{
  "motor id": "A32",
  "sensorData" {
    "motor pressure": 56,
    "motor temperature": 39
  }
}
```

以下を使用して入力を定義します。

```
{
  "inputName": "PressureInput",
  "inputDescription": "Pressure readings from a motor",
  "inputDefinition": {
    "attributes": [
      { "jsonPath": "sensorData.`motor pressure`" },
      { "jsonPath": "`motor id`" }
    ]
  }
}
```

条件表現では、バッククォートを使用してそのようなキーの値も参照する必要があります。

```
$input.PressureInput.sensorData.`motor pressure`
```

コメントされた例: を使用した HIPAA 温度制御 AWS IoT Events

次の例の JSON ファイルの一部には、コメントがインラインで含まれているため、無効な JSON になります。コメントなしのこれらの例の完全なバージョンは、例: での HIPAA 温度制御の使用 AWS IoT Events で入手できます。

この例では、次のことを実行できるサーモスタット制御モデルを実装しています。

- 複数の領域をモニタリングおよび制御するために使用できるディテクターモデルを 1 つだけ定義します。エリアごとにディテクターインスタンスが作成されます。
- 各制御領域の複数のセンサーから温度データを取り込みます。
- エリアの温度設定値を変更します。
- 各領域の操作パラメータを設定し、インスタンスの使用中にこれらのパラメータをリセットします。
- エリアからセンサーを動的に追加または削除します。
- 冷暖房単位を保護するための最小ランタイムを指定します。
- 異常なセンサー読み取り値を拒否します。
- いずれかのセンサーが特定のしきい値を超えるまたは下回る温度を報告した場合に、すぐに加熱または冷却を行う緊急設定値を定義します。
- 異常な測定値と温度スパイクを報告します。

トピック

- でのディテクターモデルの入力定義 AWS IoT Events
- AWS IoT Events ディテクターモデル定義を作成する
- BatchUpdateDetector を使用して AWS IoT Events ディテクターモデルを更新する
- での入力に BatchPutMessage を使用する AWS IoT Events
- で MQTT メッセージを取り込む AWS IoT Events
- で Amazon SNS メッセージを生成する AWS IoT Events
- で DescribeDetector API を設定する AWS IoT Events
- の AWS IoT Core ルールエンジンを使用する AWS IoT Events

でのディテクターモデルの入力定義 AWS IoT Events

いくつかの異なる領域の温度をモニタリングおよび制御するために使用できる 1 つのディテクターモデルを作成したいと思います。各エリアには、温度を報告する複数のセンサーを含めることができます。各エリアには、エリア内の温度を制御するためにオンまたはオフにできる 1 つの加熱単位と 1 つの冷却単位があります。各エリアは、1 つのディテクターインスタンスによって制御されます。

モニタリングおよび制御するさまざまな領域には、さまざまな制御パラメータを必要とするさまざまな特性がある可能性があるため、`'seedTemperatureInput'` を定義して、各領域にそれらのパラメータを提供します。これらの入力メッセージの 1 つを AWS IoT Eventsに送信すると、その領域で使用するパラメータを持つ新しいディテクターモデルインスタンスが作成されます。その入力の定義は次のとおりです。

CLI コマンド:

```
aws iotevents create-input --cli-input-json file://seedInput.json
```

ファイル: `seedInput.json`

```
{
  "inputName": "seedTemperatureInput",
  "inputDescription": "Temperature seed values.",
  "inputDefinition": {
    "attributes": [
      { "jsonPath": "areaId" },
      { "jsonPath": "desiredTemperature" },
      { "jsonPath": "allowedError" },
      { "jsonPath": "rangeHigh" },
      { "jsonPath": "rangeLow" },
      { "jsonPath": "anomalousHigh" },
      { "jsonPath": "anomalousLow" },
      { "jsonPath": "sensorCount" },
      { "jsonPath": "noDelay" }
    ]
  }
}
```

レスポンス:

```
{
```

```
    "inputConfiguration": {
        "status": "ACTIVE",
        "inputArn": "arn:aws:iotevents:us-west-2:123456789012:input/
seedTemperatureInput",
        "lastUpdateTime": 1557519620.736,
        "creationTime": 1557519620.736,
        "inputName": "seedTemperatureInput",
        "inputDescription": "Temperature seed values."
    }
}
```

メモ

- 新しいディテクターインスタンスは、メッセージで受信された一意の 'areaId' ごとに作成されます。'areaDetectorModel' 定義の 'key' フィールドを参照してください。
- 平均気温は、その地域の暖房または冷房単位が作動する前に、'allowedError' によって 'desiredTemperature' とは異なる場合があります。
- センサーが 'rangeHigh' を超える温度を報告すると、ディテクターはスパイクを報告し、すぐに冷却単位をスタートします。
- センサーが 'rangeLow' 未満の温度を報告すると、ディテクターはスパイクを報告し、すぐに加熱単位をスタートします。
- センサーが 'anomalousHigh' より上または 'anomalousLow' より下の温度を報告した場合、ディテクターは異常なセンサー読み取り値を報告しますが、報告された温度読み取り値を無視します。
- 'sensorCount' は、エリアについて報告しているセンサーの数をディテクターに通知します。ディテクターは、受信した各温度測定値に適切な重み係数を与えることにより、その領域の平均温度を計算します。このため、ディテクターは各センサーが報告する内容を追跡する必要がなく、センサーの数は必要に応じて動的に変更できます。ただし、個々のセンサーがオフラインになると、ディテクターはこれを認識したり、それを考慮したりしません。各センサーの接続状態をモニタリングするために、特別に別のディテクターモデルを作成することをお勧めします。2 つの補完的なディテクターモデルを使用すると、両方の設計が簡素化されます。
- 'noDelay' 値は true または false にすることができます。加熱または冷却単位をオンにした後は、単位の完全性を保護し、動作寿命を延ばすために、一定の最小時間オンのままにしておく必要があります。'noDelay' が false に設定されている場合、ディテクターインスタンスは、冷却単位と加熱単位をオフにする前に遅延を強制して、それらが最小時間実行されるようにします。可変値を使用してタイマーを設定できないため、遅延の秒数はディテクターモデルの定義にハードコーディングされています。

'temperatureInput' は、センサーデータをディテクターインスタンスに送信するために使用されます。

CLI コマンド:

```
aws iotevents create-input --cli-input-json file://temperatureInput.json
```

ファイル: temperatureInput.json

```
{
  "inputName": "temperatureInput",
  "inputDescription": "Temperature sensor unit data.",
  "inputDefinition": {
    "attributes": [
      { "jsonPath": "sensorId" },
      { "jsonPath": "areaId" },
      { "jsonPath": "sensorData.temperature" }
    ]
  }
}
```

レスポンス:

```
{
    "inputConfiguration": {
        "status": "ACTIVE",
        "inputArn": "arn:aws:iotevents:us-west-2:123456789012:input/temperatureInput",
        "lastUpdateTime": 1557519707.399,
        "creationTime": 1557519707.399,
        "inputName": "temperatureInput",
        "inputDescription": "Temperature sensor unit data."
    }
}
```

メモ

- 'sensorId' は、センサーを直接制御またはモニタリングするためのディテクターインスタンスの例では使用されません。これは、ディテクターインスタンスによって送信される通知に自動的に渡されます。そこから、故障しているセンサー (例えば、異常な測定値を定期的に送信するセンサーは故障しかけている可能性がある)、またはオフラインになったセンサー (デバイスのハート

ビートをモニタリングする追加のディテクターモデルへの入力として使用される場合) を特定するために使用できます。'sensorId' は、測定値が平均と定期的に異なる場合、エリア内のウォームゾーンまたはコールドゾーンを識別するのにも役立ちます。

- 'areaId' は、センサーのデータを適切なディテクターインスタンスに送信するために使用されます。ディテクターインスタンスは、メッセージで受信された一意の 'areaId' ごとに作成されます。'areaDetectorModel' 定義の 'key' フィールドを参照してください。

AWS IoT Events ディテクターモデル定義を作成する

'areaDetectorModel' の例には、コメントがインラインで含まれています。

CLI コマンド:

```
aws iotevents create-detector-model --cli-input-json file://areaDetectorModel.json
```

ファイル: areaDetectorModel.json

```
{
  "detectorModelName": "areaDetectorModel",
  "detectorModelDefinition": {
    "states": [
      {
        "stateName": "start",
        // In the 'start' state we set up the operation parameters of the new detector
 instance.
        //   We get here when the first input message arrives. If that is a
 'seedTemperatureInput'
        //   message, we save the operation parameters, then transition to the 'idle'
 state. If
        //   the first message is a 'temperatureInput', we wait here until we get a
        //   'seedTemperatureInput' input to ensure our operation parameters are set.
 We can
        //   also reenter this state using the 'BatchUpdateDetector' API. This enables
 us to
        //   reset the operation parameters without needing to delete the detector
 instance.
        "onEnter": {
          "events": [
            {
              "eventName": "prepare",
```

```
              "condition": "true",
              "actions": [
                {
                  "setVariable": {
                    // initialize 'sensorId' to an invalid value (0) until an actual
sensor reading
                    //   arrives
                    "variableName": "sensorId",
                    "value": "0"
                  }
                },
                {
                  "setVariable": {
                    // initialize 'reportedTemperature' to an invalid value (0.1) until
an actual
                    //   sensor reading arrives
                    "variableName": "reportedTemperature",
                    "value": "0.1"
                  }
                },
                {
                  "setVariable": {
                    // When using 'BatchUpdateDetector' to re-enter this state, this
variable should
                    //   be set to true.
                    "variableName": "resetMe",
                    "value": "false"
                  }
                }
              ]
            }
          ]
        },
        "onInput": {
          "transitionEvents": [
            {
              "eventName": "initialize",
              "condition": "$input.seedTemperatureInput.sensorCount > 0",
              // When a 'seedTemperatureInput' message with a valid 'sensorCount' is
received,
              //   we use it to set the operational parameters for the area to be
monitored.
              "actions": [
                {
```

```
                "setVariable": {
                  "variableName": "rangeHigh",
                  "value": "$input.seedTemperatureInput.rangeHigh"
                }
              },
              {
                "setVariable": {
                  "variableName": "rangeLow",
                  "value": "$input.seedTemperatureInput.rangeLow"
                }
              },
              {
                "setVariable": {
                  "variableName": "desiredTemperature",
                  "value": "$input.seedTemperatureInput.desiredTemperature"
                }
              },
              {
                "setVariable": {
                  // Assume we're at the desired temperature when we start.
                  "variableName": "averageTemperature",
                  "value": "$input.seedTemperatureInput.desiredTemperature"
                }
              },
              {
                "setVariable": {
                  "variableName": "allowedError",
                  "value": "$input.seedTemperatureInput.allowedError"
                }
              },
              {
                "setVariable": {
                  "variableName": "anomalousHigh",
                  "value": "$input.seedTemperatureInput.anomalousHigh"
                }
              },
              {
                "setVariable": {
                  "variableName": "anomalousLow",
                  "value": "$input.seedTemperatureInput.anomalousLow"
                }
              },
              {
                "setVariable": {
```

```
                    "variableName": "sensorCount",
                    "value": "$input.seedTemperatureInput.sensorCount"
                  }
                },
                {
                  "setVariable": {
                    "variableName": "noDelay",
                    "value": "$input.seedTemperatureInput.noDelay == true"
                  }
                }
              ],
              "nextState": "idle"
            },
            {
              "eventName": "reset",
              "condition": "($variable.resetMe == true) &&
($input.temperatureInput.sensorData.temperature < $variable.anomalousHigh &&
$input.temperatureInput.sensorData.temperature > $variable.anomalousLow)",
              // This event is triggered if we have reentered the 'start' state using
the
              //   'BatchUpdateDetector' API with 'resetMe' set to true. When we
reenter using
              //   'BatchUpdateDetector' we do not automatically continue to the 'idle'
state, but
              //   wait in 'start' until the next input message arrives. This event
enables us to
              //   transition to 'idle' on the next valid 'temperatureInput' message
that arrives.
              "actions": [
                {
                  "setVariable": {
                    "variableName": "averageTemperature",
                    "value": "((($variable.averageTemperature * ($variable.sensorCount
- 1)) + $input.temperatureInput.sensorData.temperature) / $variable.sensorCount)"
                  }
                }
              ],
              "nextState": "idle"
            }
          ]
        },
        "onExit": {
          "events": [
            {
```

```
              "eventName": "resetHeatCool",
              "condition": "true",
              // Make sure the heating and cooling units are off before entering
'idle'.
              "actions": [
                {
                  "sns": {
                    "targetArn": "arn:aws:sns:us-west-2:123456789012:heatOff"
                  }
                },
                {
                  "sns": {
                    "targetArn": "arn:aws:sns:us-west-2:123456789012:coolOff"
                  }
                },
                {
                  "iotTopicPublish": {
                    "mqttTopic": "hvac/Heating/Off"
                  }
                },
                {
                  "iotTopicPublish": {
                    "mqttTopic": "hvac/Cooling/Off"
                  }
                }
              ]
            }
          ]
        }
      },


      {
        "stateName": "idle",
        "onInput": {
          "events": [
            {
              "eventName": "whatWasInput",
              "condition": "true",
              // By storing the 'sensorId' and the 'temperature' in variables, we make
them
              //   available in any messages we send out to report anomalies, spikes,
or just
              //   if needed for debugging.
```

```
              "actions": [
                {
                  "setVariable": {
                    "variableName": "sensorId",
                    "value": "$input.temperatureInput.sensorId"
                  }
                },
                {
                  "setVariable": {
                    "variableName": "reportedTemperature",
                    "value": "$input.temperatureInput.sensorData.temperature"
                  }
                }
              ]
            },
            {
              "eventName": "changeDesired",
              "condition": "$input.seedTemperatureInput.desiredTemperature !=
$variable.desiredTemperature",
              // This event enables us to change the desired temperature at any time by
sending a
              //   'seedTemperatureInput' message. But note that other operational
parameters are not
              //   read or changed.
              "actions": [
                {
                  "setVariable": {
                    "variableName": "desiredTemperature",
                    "value": "$input.seedTemperatureInput.desiredTemperature"
                  }
                }
              ]
            },
            {
              "eventName": "calculateAverage",
              "condition": "$input.temperatureInput.sensorData.temperature <
$variable.anomalousHigh && $input.temperatureInput.sensorData.temperature >
$variable.anomalousLow",
              // If a valid temperature reading arrives, we use it to update the
average temperature.
              //   For simplicity, we assume our sensors will be sending updates at
about the same rate,
              //   so we can calculate an approximate average by giving equal weight to
each reading we receive.
```

```
              "actions": [
                {
                  "setVariable": {
                    "variableName": "averageTemperature",
                    "value": "((($variable.averageTemperature * ($variable.sensorCount
- 1)) + $input.temperatureInput.sensorData.temperature) / $variable.sensorCount)"
                  }
                }
              ]
            }
          ],
          "transitionEvents": [
            {
              "eventName": "anomalousInputArrived",
              "condition": "$input.temperatureInput.sensorData.temperature >=
$variable.anomalousHigh || $input.temperatureInput.sensorData.temperature <=
$variable.anomalousLow",
              // When an anomalous reading arrives, send an MQTT message, but stay in
the current state.
              "actions": [
                {
                  "iotTopicPublish": {
                    "mqttTopic": "temperatureSensor/anomaly"
                  }
                }
              ],
              "nextState": "idle"
            },

            {
              "eventName": "highTemperatureSpike",
              "condition": "$input.temperatureInput.sensorData.temperature >
$variable.rangeHigh",
              // When even a single temperature reading arrives that is above the
'rangeHigh', take
              //   emergency action to begin cooling, and report a high temperature
spike.
              "actions": [
                {
                  "iotTopicPublish": {
                    "mqttTopic": "temperatureSensor/spike"
                  }
                },
                {
```

```
                "sns": {
                  "targetArn": "arn:aws:sns:us-west-2:123456789012:coolOn"
                }
              },
              {
                "iotTopicPublish": {
                  "mqttTopic": "hvac/Cooling/On"
                }
              },
              {
                "setVariable": {
                  // This is necessary because we want to set a timer to delay the
shutoff
                  //   of a cooling/heating unit, but we only want to set the timer
when we
                  //   enter that new state initially.
                  "variableName": "enteringNewState",
                  "value": "true"
                }
              }
            ],
            "nextState": "cooling"
          },

          {
            "eventName": "lowTemperatureSpike",
            "condition": "$input.temperatureInput.sensorData.temperature <
$variable.rangeLow",
            // When even a single temperature reading arrives that is below the
'rangeLow', take
            //   emergency action to begin heating, and report a low-temperature
spike.
            "actions": [
              {
                "iotTopicPublish": {
                  "mqttTopic": "temperatureSensor/spike"
                }
              },
              {
                "sns": {
                  "targetArn": "arn:aws:sns:us-west-2:123456789012:heatOn"
                }
              },
              {
```

```
                "iotTopicPublish": {
                  "mqttTopic": "hvac/Heating/On"
                }
              },
              {
                "setVariable": {
                  "variableName": "enteringNewState",
                  "value": "true"
                }
              }
            ],
            "nextState": "heating"
          },

          {
            "eventName": "highTemperatureThreshold",
            "condition": "((((($variable.averageTemperature * ($variable.sensorCount
- 1)) + $input.temperatureInput.sensorData.temperature) / $variable.sensorCount) >
($variable.desiredTemperature + $variable.allowedError))",
            // When the average temperature is above the desired temperature plus the
allowed error factor,
            //   it is time to start cooling. Note that we calculate the average
temperature here again
            //   because the value stored in the 'averageTemperature' variable is not
yet available for use
            //   in our condition.
            "actions": [
              {
                "sns": {
                  "targetArn": "arn:aws:sns:us-west-2:123456789012:coolOn"
                }
              },
              {
                "iotTopicPublish": {
                  "mqttTopic": "hvac/Cooling/On"
                }
              },
              {
                "setVariable": {
                  "variableName": "enteringNewState",
                  "value": "true"
                }
              }
            ],
```

```
              "nextState": "cooling"
            },

            {
              "eventName": "lowTemperatureThreshold",
              "condition": "((((($variable.averageTemperature * ($variable.sensorCount
- 1)) + $input.temperatureInput.sensorData.temperature) / $variable.sensorCount) <
($variable.desiredTemperature - $variable.allowedError))",
              // When the average temperature is below the desired temperature minus
the allowed error factor,
              //   it is time to start heating. Note that we calculate the average
temperature here again
              //   because the value stored in the 'averageTemperature' variable is not
yet available for use
              //   in our condition.
              "actions": [
                {
                  "sns": {
                    "targetArn": "arn:aws:sns:us-west-2:123456789012:heatOn"
                  }
                },
                {
                  "iotTopicPublish": {
                    "mqttTopic": "hvac/Heating/On"
                  }
                },
                {
                  "setVariable": {
                    "variableName": "enteringNewState",
                    "value": "true"
                  }
                }
              ],
              "nextState": "heating"
            }
          ]
        }
      },


      {
        "stateName": "cooling",
        "onEnter": {
          "events": [
```

```
            {
              "eventName": "delay",
              "condition": "!$variable.noDelay && $variable.enteringNewState",
              // If the operational parameters specify that there should be a minimum
time that the
              //   heating and cooling units should be run before being shut off again,
we set
              //   a timer to ensure the proper operation here.
              "actions": [
                {
                  "setTimer": {
                    "timerName": "coolingTimer",
                    "seconds": 180
                  }
                },
                {
                  "setVariable": {
                    // We use this 'goodToGo' variable to store the status of the timer
expiration
                    //   for use in conditions that also use input variable values. If
                    //   'timeout()' is used in such mixed conditionals, its value is
lost.
                    "variableName": "goodToGo",
                    "value": "false"
                  }
                }
              ]
            },
            {
              "eventName": "dontDelay",
              "condition": "$variable.noDelay == true",
              // If the heating/cooling unit shutoff delay is not used, no need to
wait.
              "actions": [
                {
                  "setVariable": {
                    "variableName": "goodToGo",
                    "value": "true"
                  }
                }
              ]
            },
            {
              "eventName": "beenHere",
```

```
              "condition": "true",
              "actions": [
                {
                  "setVariable": {
                    "variableName": "enteringNewState",
                    "value": "false"
                  }
                }
              ]
            }
          ]
        },

        "onInput": {
          "events": [
            // These are events that occur when an input is received (if the condition
is
            //   satisfied), but don't cause a transition to another state.
            {
              "eventName": "whatWasInput",
              "condition": "true",
              "actions": [
                {
                  "setVariable": {
                    "variableName": "sensorId",
                    "value": "$input.temperatureInput.sensorId"
                  }
                },
                {
                  "setVariable": {
                    "variableName": "reportedTemperature",
                    "value": "$input.temperatureInput.sensorData.temperature"
                  }
                }
              ]
            },
            {
              "eventName": "changeDesired",
              "condition": "$input.seedTemperatureInput.desiredTemperature !=
$variable.desiredTemperature",
              "actions": [
                {
                  "setVariable": {
                    "variableName": "desiredTemperature",
```

```
                    "value": "$input.seedTemperatureInput.desiredTemperature"
                  }
                }
              ]
            },
            {
              "eventName": "calculateAverage",
              "condition": "$input.temperatureInput.sensorData.temperature <
$variable.anomalousHigh && $input.temperatureInput.sensorData.temperature >
$variable.anomalousLow",
              "actions": [
                {
                  "setVariable": {
                    "variableName": "averageTemperature",
                    "value": "((($variable.averageTemperature * ($variable.sensorCount
- 1)) + $input.temperatureInput.sensorData.temperature) / $variable.sensorCount)"
                  }
                }
              ]
            },
            {
              "eventName": "areWeThereYet",
              "condition": "(timeout(\"coolingTimer\"))",
              "actions": [
                {
                  "setVariable": {
                    "variableName": "goodToGo",
                    "value": "true"
                  }
                }
              ]
            }
          ],
          "transitionEvents": [
            // Note that some tests of temperature values (for example, the test for an
anomalous value)
            //   must be placed here in the 'transitionEvents' because they work
together with the tests
            //   in the other conditions to ensure that we implement the proper
"if..elseif..else" logic.
            //   But each transition event must have a destination state ('nextState'),
and even if that
            //   is actually the current state, the "onEnter" events for this state
will be executed again.
```

```
            //   This is the reason for the 'enteringNewState' variable and related.
            {
              "eventName": "anomalousInputArrived",
              "condition": "$input.temperatureInput.sensorData.temperature >=
$variable.anomalousHigh || $input.temperatureInput.sensorData.temperature <=
$variable.anomalousLow",
              "actions": [
                {
                  "iotTopicPublish": {
                    "mqttTopic": "temperatureSensor/anomaly"
                  }
                }
              ],
              "nextState": "cooling"
            },

            {
              "eventName": "highTemperatureSpike",
              "condition": "$input.temperatureInput.sensorData.temperature >
$variable.rangeHigh",
              "actions": [
                {
                  "iotTopicPublish": {
                    "mqttTopic": "temperatureSensor/spike"
                  }
                }
              ],
              "nextState": "cooling"
            },

            {
              "eventName": "lowTemperatureSpike",
              "condition": "$input.temperatureInput.sensorData.temperature <
$variable.rangeLow",
              "actions": [
                {
                  "iotTopicPublish": {
                    "mqttTopic": "temperatureSensor/spike"
                  }
                },
                {
                  "sns": {
                    "targetArn": "arn:aws:sns:us-west-2:123456789012:coolOff"
                  }
```

```
                },
                {
                  "sns": {
                    "targetArn": "arn:aws:sns:us-west-2:123456789012:heatOn"
                  }
                },
                {
                  "iotTopicPublish": {
                    "mqttTopic": "hvac/Cooling/Off"
                  }
                },
                {
                  "iotTopicPublish": {
                    "mqttTopic": "hvac/Heating/On"
                  }
                },
                {
                  "setVariable": {
                    "variableName": "enteringNewState",
                    "value": "true"
                  }
                }
              ],
              "nextState": "heating"
            },

            {
              "eventName": "desiredTemperature",
              "condition": "((((($variable.averageTemperature * ($variable.sensorCount
- 1)) + $input.temperatureInput.sensorData.temperature) / $variable.sensorCount) <=
($variable.desiredTemperature - $variable.allowedError)) && $variable.goodToGo ==
true",
              "actions": [
                {
                  "sns": {
                    "targetArn": "arn:aws:sns:us-west-2:123456789012:coolOff"
                  }
                },
                {
                  "iotTopicPublish": {
                    "mqttTopic": "hvac/Cooling/Off"
                  }
                }
              ],
```

```
            "nextState": "idle"
          }
        ]
      }
    },


    {
      "stateName": "heating",
      "onEnter": {
        "events": [
          {
            "eventName": "delay",
            "condition": "!$variable.noDelay && $variable.enteringNewState",
            "actions": [
              {
                "setTimer": {
                  "timerName": "heatingTimer",
                  "seconds": 120
                }
              },
              {
                "setVariable": {
                  "variableName": "goodToGo",
                  "value": "false"
                }
              }
            ]
          },
          {
            "eventName": "dontDelay",
            "condition": "$variable.noDelay == true",
            "actions": [
              {
                "setVariable": {
                  "variableName": "goodToGo",
                  "value": "true"
                }
              }
            ]
          },
          {
            "eventName": "beenHere",
            "condition": "true",
```

```
              "actions": [
                {
                  "setVariable": {
                    "variableName": "enteringNewState",
                    "value": "false"
                  }
                }
              ]
            }
          ]
        },

        "onInput": {
          "events": [
            {
              "eventName": "whatWasInput",
              "condition": "true",
              "actions": [
                {
                  "setVariable": {
                    "variableName": "sensorId",
                    "value": "$input.temperatureInput.sensorId"
                  }
                },
                {
                  "setVariable": {
                    "variableName": "reportedTemperature",
                    "value": "$input.temperatureInput.sensorData.temperature"
                  }
                }
              ]
            },
            {
              "eventName": "changeDesired",
              "condition": "$input.seedTemperatureInput.desiredTemperature !=
 $variable.desiredTemperature",
              "actions": [
                {
                  "setVariable": {
                    "variableName": "desiredTemperature",
                    "value": "$input.seedTemperatureInput.desiredTemperature"
                  }
                }
              ]
```

```
            },
            {
              "eventName": "calculateAverage",
              "condition": "$input.temperatureInput.sensorData.temperature <
$variable.anomalousHigh && $input.temperatureInput.sensorData.temperature >
$variable.anomalousLow",
              "actions": [
                {
                  "setVariable": {
                    "variableName": "averageTemperature",
                    "value": "((($variable.averageTemperature * ($variable.sensorCount
- 1)) + $input.temperatureInput.sensorData.temperature) / $variable.sensorCount)"
                  }
                }
              ]
            },
            {
              "eventName": "areWeThereYet",
              "condition": "(timeout(\"heatingTimer\"))",
              "actions": [
                {
                  "setVariable": {
                    "variableName": "goodToGo",
                    "value": "true"
                  }
                }
              ]
            }
          ],
          "transitionEvents": [
            {
              "eventName": "anomalousInputArrived",
              "condition": "$input.temperatureInput.sensorData.temperature >=
$variable.anomalousHigh || $input.temperatureInput.sensorData.temperature <=
$variable.anomalousLow",
              "actions": [
                {
                  "iotTopicPublish": {
                    "mqttTopic": "temperatureSensor/anomaly"
                  }
                }
              ],
              "nextState": "heating"
            },
```

```
            {
              "eventName": "highTemperatureSpike",
              "condition": "$input.temperatureInput.sensorData.temperature >
 $variable.rangeHigh",
              "actions": [
                {
                  "iotTopicPublish": {
                    "mqttTopic": "temperatureSensor/spike"
                  }
                },
                {
                  "sns": {
                    "targetArn": "arn:aws:sns:us-west-2:123456789012:heatOff"
                  }
                },
                {
                  "sns": {
                    "targetArn": "arn:aws:sns:us-west-2:123456789012:coolOn"
                  }
                },
                {
                  "iotTopicPublish": {
                    "mqttTopic": "hvac/Heating/Off"
                  }
                },
                {
                  "iotTopicPublish": {
                    "mqttTopic": "hvac/Cooling/On"
                  }
                },
                {
                  "setVariable": {
                    "variableName": "enteringNewState",
                    "value": "true"
                  }
                }
              ],
              "nextState": "cooling"
            },

            {
              "eventName": "lowTemperatureSpike",
```

```
              "condition": "$input.temperatureInput.sensorData.temperature <
 $variable.rangeLow",
              "actions": [
                {
                  "iotTopicPublish": {
                    "mqttTopic": "temperatureSensor/spike"
                  }
                }
              ],
              "nextState": "heating"
            },

            {
              "eventName": "desiredTemperature",
              "condition": "((((($variable.averageTemperature * ($variable.sensorCount
 - 1)) + $input.temperatureInput.sensorData.temperature) / $variable.sensorCount) >=
 ($variable.desiredTemperature + $variable.allowedError)) && $variable.goodToGo ==
 true",
              "actions": [
                {
                  "sns": {
                    "targetArn": "arn:aws:sns:us-west-2:123456789012:heatOff"
                  }
                },
                {
                  "iotTopicPublish": {
                    "mqttTopic": "hvac/Heating/Off"
                  }
                }
              ],
              "nextState": "idle"
            }
          ]
        }
      }

    ],

    "initialStateName": "start"
  },
  "key": "areaId",
  "roleArn": "arn:aws:iam::123456789012:role/IoTEventsRole"
}
```

レスポンス:

```
{
    "detectorModelConfiguration": {
        "status": "ACTIVATING",
        "lastUpdateTime": 1557523491.168,
        "roleArn": "arn:aws:iam::123456789012:role/IoTEventsRole",
        "creationTime": 1557523491.168,
        "detectorModelArn": "arn:aws:iotevents:us-west-2:123456789012:detectorModel/
areaDetectorModel",
        "key": "areaId",
        "detectorModelName": "areaDetectorModel",
        "detectorModelVersion": "1"
    }
}
```

BatchUpdateDetector を使用して AWS IoT Events ディテクターモデルを更新する

`BatchUpdateDetector` オペレーションを使用して、ディテクターインスタンスをタイマーや可変値などの既知の状態にすることができます。次の例では、`BatchUpdateDetector` オペレーションにより、温度のモニタリングと制御が行われているエリアの操作パラメータがリセットされます。このオペレーションにより、ディテクターモデルを削除、再作成、または更新することなく、これを行うことができます。

CLI コマンド:

```
aws iotevents-data batch-update-detector --cli-input-json file://areaDM.BUD.json
```

ファイル: `areaDM.BUD.json`

```
{
  "detectors": [
    {
      "messageId": "0001",
      "detectorModelName": "areaDetectorModel",
      "keyValue": "Area51",
      "state": {
        "stateName": "start",
        "variables": [
```

```
            {
              "name": "desiredTemperature",
              "value": "22"
            },
            {
              "name": "averageTemperature",
              "value": "22"
            },
            {
              "name": "allowedError",
              "value": "1.0"
            },
            {
              "name": "rangeHigh",
              "value": "30.0"
            },
            {
              "name": "rangeLow",
              "value": "15.0"
            },
            {
              "name": "anomalousHigh",
              "value": "60.0"
            },
            {
              "name": "anomalousLow",
              "value": "0.0"
            },
            {
              "name": "sensorCount",
              "value": "12"
            },
            {
              "name": "noDelay",
              "value": "true"
            },
            {
              "name": "goodToGo",
              "value": "true"
            },
            {
              "name": "sensorId",
              "value": "0"
            },
```

```
          {
            "name": "reportedTemperature",
            "value": "0.1"
          },
          {
            "name": "resetMe",
            // When 'resetMe' is true, our detector model knows that we have reentered
 the 'start' state
            //   to reset operational parameters, and will allow the next valid
 temperature sensor
            //   reading to cause the transition to the 'idle' state.
            "value": "true"
          }
        ],
        "timers": [
        ]
      }
    }
  ]
}
```

レスポンス:

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

での入力に BatchPutMessage を使用する AWS IoT Events

Example 1

`BatchPutMessage` オペレーションを使用して、温度制御およびモニタリング下の特定の領域の操作パラメータを設定する `"seedTemperatureInput"` メッセージを送信します。新しい を持つ によって受信 AWS IoT Events されたメッセージ`"areaId"`により、新しいディテクターインスタンスが作成されます。ただし、新しいディテクターインスタンスは状態を `"idle"` に変更せず、新しい領域の `"seedTemperatureInput"` メッセージが受信されるまで、温度のモニタリングと暖房または冷房単位の制御を開始します。

CLI コマンド:

```
aws iotevents-data batch-put-message --cli-input-json file://seedExample.json --cli-
binary-format raw-in-base64-out
```

ファイル: `seedExample.json`

```
{
  "messages": [
    {
      "messageId": "00001",
      "inputName": "seedTemperatureInput",
      "payload": "{\"areaId\": \"Area51\", \"desiredTemperature\": 20.0, \"allowedError
\": 0.7, \"rangeHigh\": 30.0, \"rangeLow\": 15.0, \"anomalousHigh\": 60.0,
 \"anomalousLow\": 0.0, \"sensorCount\": 10, \"noDelay\": false}"
    }
  ]
}
```

レスポンス:

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

Example

2

`BatchPutMessage` オペレーションを使用して `"temperatureInput"` メッセージを送信し、特定の制御およびモニタリング領域にあるセンサーの温度センサーデータを報告します。

CLI コマンド:

```
aws iotevents-data batch-put-message --cli-input-json file://temperatureExample.json --
cli-binary-format raw-in-base64-out
```

ファイル: `temperatureExample.json`

```
{
  "messages": [
    {
      "messageId": "00005",
```

```
      "inputName": "temperatureInput",
      "payload": "{\"sensorId\": \"05\", \"areaId\": \"Area51\", \"sensorData\":
 {\"temperature\": 23.12} }"
    }
  ]
}
```

レスポンス:

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

Example 3

`BatchPutMessage` オペレーションを使用して `"seedTemperatureInput"` メッセージを送信し、特定の領域の目的の温度の値を変更します。

CLI コマンド:

```
aws iotevents-data batch-put-message --cli-input-json file://seedSetDesiredTemp.json --
cli-binary-format raw-in-base64-out
```

ファイル: `seedSetDesiredTemp.json`

```
{
  "messages": [
    {
      "messageId": "00001",
      "inputName": "seedTemperatureInput",
      "payload": "{\"areaId\": \"Area51\", \"desiredTemperature\": 23.0}"
    }
  ]
}
```

レスポンス:

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

で MQTT メッセージを取り込む AWS IoT Events

センサーコンピューティングリソースは "BatchPutMessage" API を使用できませんが、軽量 MQTT クライアントを使用して AWS IoT Core メッセージブローカーにデータを送信できる場合は、 AWS IoT Core トピックルールを作成してメッセージデータを AWS IoT Events 入力にリダイレクトできます。以下は、MQTT AWS IoT Events トピックから "areaId"および "sensorId"入力フィールドを取得し、メッセージペイロード"sensorData.temperature"フィールドから "temp"フィールドを取得し、このデータを に取り込む トピックルールの定義です AWS IoT Events "temperatureInput"。

CLI コマンド:

```
aws iot create-topic-rule --cli-input-json file://temperatureTopicRule.json
```

ファイル: seedSetDesiredTemp.json

```
{
  "ruleName": "temperatureTopicRule",
  "topicRulePayload": {
    "sql": "SELECT topic(3) as areaId, topic(4) as sensorId, temp as
 sensorData.temperature FROM 'update/temperature/#'",
    "description": "Ingest temperature sensor messages into IoT Events",
    "actions": [
      {
        "iotEvents": {
          "inputName": "temperatureInput",
          "roleArn": "arn:aws:iam::123456789012:role/service-role/anotheRole"
        }
      }
    ],
    "ruleDisabled": false,
    "awsIotSqlVersion": "2016-03-23"
  }
}
```

レスポンス: [なし]

センサーがトピック "update/temperature/Area51/03" に関するメッセージを次のペイロードで送信した場合。

```
{ "temp": 24.5 }
```

これにより、次の "BatchPutMessage" API コールが行われたか AWS IoT Events のようにデータが に取り込まれます。

```
aws iotevents-data batch-put-message --cli-input-json file://spoofExample.json --cli-
binary-format raw-in-base64-out
```

ファイル: spoofExample.json

```
{
  "messages": [
    {
      "messageId": "54321",
      "inputName": "temperatureInput",
      "payload": "{\"sensorId\": \"03\", \"areaId\": \"Area51\", \"sensorData\":
 {\"temperature\": 24.5} }"
    }
  ]
}
```

で Amazon SNS メッセージを生成する AWS IoT Events

以下は、"Area51" ディテクターインスタンスによって生成された SNS メッセージの例です。

AWS IoT Events は Amazon SNS と統合して、検出されたイベントに基づいて通知を生成および発行できます。このセクションでは、ディ AWS IoT Events テクターインスタンス、特に「Area51」ディテクターが Amazon SNS メッセージを生成する方法を示します。これらの例は、ディ AWS IoT Events テクター内のさまざまな状態やイベントによってトリガーされる Amazon SNS 通知の構造と内容を示し、リアルタイムのアラートと通信のために Amazon SNS AWS IoT Events と組み合わせる能力を示しています。

```
Heating system off command> {
  "eventTime":1557520274729,
  "payload":{
    "actionExecutionId":"f3159081-bac3-38a4-96f7-74af0940d0a4",
    "detector":{

 "detectorModelName":"areaDetectorModel","keyValue":"Area51","detectorModelVersion":"1"},"event
{"inputName":"seedTemperatureInput","messageId":"00001","triggerType":"Message"},"state":
```

```
{"stateName":"start","variables":
{"sensorCount":10,"rangeHigh":30.0,"resetMe":false,"enteringNewState":true,"averageTemperature"
{}}},"eventName":"resetHeatCool"}
```

```
Cooling system off command> {"eventTime":1557520274729,"payload":
{"actionExecutionId":"98f6a1b5-8f40-3cdb-9256-93afd4d66192","detector":
{"detectorModelName":"areaDetectorModel","keyValue":"Area51","detectorModelVersion":"1"},"event
{"inputName":"seedTemperatureInput","messageId":"00001","triggerType":"Message"},"state":
{"stateName":"start","variables":
{"sensorCount":10,"rangeHigh":30.0,"resetMe":false,"enteringNewState":true,"averageTemperature"
{}}},"eventName":"resetHeatCool"}
```

で DescribeDetector API を設定する AWS IoT Events

の `DescribeDetector` API AWS IoT Events を使用すると、特定のディテクターインスタンスに関する詳細情報を取得できます。このオペレーションは、ディテクターの現在の状態、変数値、アクティブなタイマーに関するインサイトを提供します。この API を使用すると、 AWS IoT Events ディテクターのリアルタイムステータスをモニタリングできるため、IoT イベント処理ワークフローのデバッグ、分析、管理が容易になります。

CLI コマンド:

```
aws iotevents-data describe-detector --detector-model-name areaDetectorModel --key-
value Area51
```

レスポンス:

```
{
    "detector": {
        "lastUpdateTime": 1557521572.216,
        "creationTime": 1557520274.405,
        "state": {
            "variables": [
                {
                    "name": "resetMe",
                    "value": "false"
                },
                {
                    "name": "rangeLow",
```

```
                "value": "15.0"
            },
            {
                "name": "noDelay",
                "value": "false"
            },
            {
                "name": "desiredTemperature",
                "value": "20.0"
            },
            {
                "name": "anomalousLow",
                "value": "0.0"
            },
            {
                "name": "sensorId",
                "value": "\"01\""
            },
            {
                "name": "sensorCount",
                "value": "10"
            },
            {
                "name": "rangeHigh",
                "value": "30.0"
            },
            {
                "name": "enteringNewState",
                "value": "false"
            },
            {
                "name": "averageTemperature",
                "value": "19.572"
            },
            {
                "name": "allowedError",
                "value": "0.7"
            },
            {
                "name": "anomalousHigh",
                "value": "60.0"
            },
            {
                "name": "reportedTemperature",
```

```
                    "value": "15.72"
                },
                {
                    "name": "goodToGo",
                    "value": "false"
                }
            ],
            "stateName": "idle",
            "timers": [
                {
                    "timestamp": 1557520454.0,
                    "name": "idleTimer"
                }
            ]
        },
        "keyValue": "Area51",
        "detectorModelName": "areaDetectorModel",
        "detectorModelVersion": "1"
    }
}
```

の AWS IoT Core ルールエンジンを使用する AWS IoT Events

次のルールは、 AWS IoT Core MQTT メッセージをシャドウ更新リクエストメッセージとして再発行します。ここでは、ディテクターモデルによって制御される各エリアの加熱単位と冷却単位に対して AWS IoT Core モノが定義されていることを前提としています。この例では、`"Area51HeatingUnit"` および `"Area51CoolingUnit"` という名前のものを定義しました。

CLI コマンド:

```
aws iot create-topic-rule --cli-input-json file://ADMShadowCoolOffRule.json
```

ファイル`ADMShadowCoolOffRule.json`

```
{
  "ruleName": "ADMShadowCoolOff",
  "topicRulePayload": {
    "sql": "SELECT topic(3) as state.desired.command FROM 'hvac/Cooling/Off'",
    "description": "areaDetectorModel mqtt topic publish to cooling unit shadow
 request",
    "ruleDisabled": false,
```

```
    "awsIotSqlVersion": "2016-03-23",
    "actions": [
      {
        "republish": {
          "topic": "$$aws/things/${payload.detector.keyValue}CoolingUnit/shadow/
update",
          "roleArn": "arn:aws:iam::123456789012:role/service-role/ADMShadowRole"
        }
      }
    ]
  }
}
```

レスポンス: [空]

CLI コマンド:

```
aws iot create-topic-rule --cli-input-json file://ADMShadowCoolOnRule.json
```

ファイル`ADMShadowCoolOnRule.json`

```
{
  "ruleName": "ADMShadowCoolOn",
  "topicRulePayload": {
    "sql": "SELECT topic(3) as state.desired.command FROM 'hvac/Cooling/On'",
    "description": "areaDetectorModel mqtt topic publish to cooling unit shadow
 request",
    "ruleDisabled": false,
    "awsIotSqlVersion": "2016-03-23",
    "actions": [
      {
        "republish": {
          "topic": "$$aws/things/${payload.detector.keyValue}CoolingUnit/shadow/
update",
          "roleArn": "arn:aws:iam::123456789012:role/service-role/ADMShadowRole"
        }
      }
    ]
  }
}
```

レスポンス: [空]

CLI コマンド:

```
aws iot create-topic-rule --cli-input-json file://ADMShadowHeatOffRule.json
```

ファイルADMShadowHeatOffRule.json

```
{
  "ruleName": "ADMShadowHeatOff",
  "topicRulePayload": {
    "sql": "SELECT topic(3) as state.desired.command FROM 'hvac/Heating/Off'",
    "description": "areaDetectorModel mqtt topic publish to heating unit shadow
 request",
    "ruleDisabled": false,
    "awsIotSqlVersion": "2016-03-23",
    "actions": [
      {
        "republish": {
          "topic": "$$aws/things/${payload.detector.keyValue}HeatingUnit/shadow/
update",
          "roleArn": "arn:aws:iam::123456789012:role/service-role/ADMShadowRole"
        }
      }
    ]
  }
}
```

レスポンス: [空]

CLI コマンド:

```
aws iot create-topic-rule --cli-input-json file://ADMShadowHeatOnRule.json
```

ファイルADMShadowHeatOnRule.json

```
{
  "ruleName": "ADMShadowHeatOn",
  "topicRulePayload": {
    "sql": "SELECT topic(3) as state.desired.command FROM 'hvac/Heating/On'",
    "description": "areaDetectorModel mqtt topic publish to heating unit shadow
 request",
```

```
    "ruleDisabled": false,
    "awsIotSqlVersion": "2016-03-23",
    "actions": [
      {
        "republish": {
          "topic": "$$aws/things/${payload.detector.keyValue}HeatingUnit/shadow/
update",
          "roleArn": "arn:aws:iam::123456789012:role/service-role/ADMShadowRole"
        }
      }
    ]
  }
}
```

レスポンス: [空]

でデータを受信してアクションをトリガーするためにサポートされているアクション AWS IoT Events

AWS IoT Events は、指定されたイベントまたは移行イベントを検出したときにアクションをトリガーできます。タイマーの使用や可変の設定、他の AWS リソースへのデータ送信などの組み込みアクションを定義することができます。これらのアクションを設定およびカスタマイズして、さまざまな IoT イベントへの自動応答を作成する方法について説明します。

> **Note**
>
> ディテクタモデルでアクションを定義するとき、パラメータに文字列データ型の表現を使用することができます。詳細については、「表現」を参照してください。

AWS IoT Events では、タイマーを使用したり、変数を設定したりできる以下のアクションがサポートされています。

- `setTimer` は、タイマーを作成します。
- `resetTimer` は、タイマーをリセットします。
- `clearTimer` は、タイマーを削除します。
- `setVariable` は、可変を作成します。

AWS IoT Events では、 AWS サービスを操作できる以下のアクションがサポートされています。

- `iotTopicPublish` は、MQTT トピックにメッセージを発行します。
- `iotEvents` は、 AWS IoT Events に対してデータを入力値として送信します。
- `iotSiteWise` は、 AWS IoT SiteWise中のアセットプロパティにデータを送信します。
- `dynamoDB` は、Amazon DynamoDB テーブルにデータを送信します。
- `dynamoDBv2` は、Amazon DynamoDB テーブルにデータを送信します。
- `firehose` Amazon Data Firehose ストリームにデータを送信する 。
- `lambda` は、 AWS Lambda 関数を呼び出します。
- `sns` は、データをプッシュ通知として送信します。
- `sqs` は、Amazon SQS キューにデータを送信します。

AWS IoT Events 組み込みタイマーと変数アクションを使用する

AWS IoT Events では、タイマーを使用したり、変数を設定したりできる以下のアクションがサポートされています。

- `setTimer` は、タイマーを作成します。
- `resetTimer` は、タイマーをリセットします。
- `clearTimer` は、タイマーを削除します。
- `setVariable`は、可変を作成します。

タイマーアクションの設定

Set timer action

`setTimer` アクションを使用すると、持続時間が秒単位のタイマーを作成できます。

More information (2)

タイマーを作成するときは、次のパラメータを指定する必要があります。

`timerName`

タイマーの名前。

`durationExpression`

(オプション) タイマーの期間 (秒単位)。

期間表現の評価結果は、最も近い整数に切り捨てられます。例えば、タイマーを 60,99 秒に設定すると、表現の評価結果は 60 秒になります。

詳細については、「AWS IoT Events API リファレンス」の「SetTimerAction」を参照してください。

タイマーアクションのリセット

Reset timer action

`resetTimer` アクションを使用すると、以前に評価された期間表現の結果にタイマーを設定できます。

More information (1)

タイマーをリセットしたら、以下ののパラメータを指定する必要があります。

timerName

タイマーの名前。

AWS IoT Events タイマーをリセットしても、 は期間式を再評価しません。

詳細については、「AWS IoT Events API リファレンス」の「ResetTimerAction」を参照してください。

タイマーアクションをクリア

Clear timer action

`clearTimer`アクションを使用すると、既存のタイマーを削除できます。

More information (1)

タイマーをリセットしたら、以下ののパラメータを指定する必要があります。

timerName

タイマーの名前。

詳細については、「AWS IoT Events API リファレンス」の「ClearTimerAction」を参照してください。

可変アクションの設定

Set variable action

`setVariable` アクションを使用すると、指定した値で可変を作成できます。

More information (2)

可変を作成するときは、次のパラメータを指定する必要があります。

variableName

可変の名前。

value

可変の値

詳細については、「AWS IoT Events API リファレンス」の「SetVariableAction」を参照してください。

AWS IoT Events 他の AWS サービスの使用

AWS IoT Events では、 AWS サービスを操作できる以下のアクションがサポートされています。

- `iotTopicPublish` は、MQTT トピックにメッセージを発行します。
- `iotEvents` は、 AWS IoT Events に対してデータを入力値として送信します。
- `iotSiteWise` は、 AWS IoT SiteWise中のアセットプロパティにデータを送信します。
- `dynamoDB` は、Amazon DynamoDB テーブルにデータを送信します。
- `dynamoDBv2` は、Amazon DynamoDB テーブルにデータを送信します。
- `firehose` Amazon Data Firehose ストリームにデータを送信する 。
- `lambda` は、 AWS Lambda 関数を呼び出します。
- `sns` は、データをプッシュ通知として送信します。
- `sqs` は、Amazon SQS キューにデータを送信します。

> **Important**
>
> - AWS IoT Events と サービスの両方が連携するには、 AWS 同じ AWS リージョンを選択する必要があります。サポートされているリージョンのリストについては、「Amazon Web Services 全般のリファレンス」の「AWS IoT Events の エンドポイントとクォータ」を参照してください。
> - AWS IoT Events アクションの他の AWS リソースを作成するときは、同じ AWS リージョンを使用する必要があります。 AWS リージョンを切り替えると、 AWS リソースへのアクセスに問題がある可能性があります。

デフォルトでは、 は任意のアクションの標準ペイロードを JSON で AWS IoT Events 生成します。このアクションペイロードには、ディテクターモデルインスタンスと、アクションをトリガーしたイベントに関する情報を記述した、属性と値のペアがすべて含まれています。アクションペイロードを設定するには、 コンテント表現を使用します。詳細については、「AWS IoT Events API リファレンス」の「イベントデータをフィルタリング、変換、処理する式」および「ペイロード」データ型を参照してください。

AWS IoT Core

IoT topic publish action

AWS IoT Core アクションを使用すると、メッセージブローカーを介して MQTT AWS IoT メッセージを発行できます。サポートされているリージョンのリストについては、「Amazon Web Services 全般のリファレンス」の「AWS IoT Core の エンドポイントとクォータ」を参照してください。

AWS IoT メッセージブローカーは、発行 AWS IoT 側クライアントからサブスクライブ側クライアントにメッセージを送信することで、クライアントを接続します。詳細については、「 AWS IoT デベロッパーガイド」の「デバイス通信プロトコル」を参照してください。

More information (2)

MQTT メッセージを発行するときは、以下のパラメータを指定する必要があります。

`mqttTopic`

メッセージを受信する MQTT トピック。

ディテクタモデルで作成された可変または入力値を使用して、ランタイムに MQTT トピック名を動的に定義できます。

`payload`

(オプション) デフォルトのペイロードには、ディテクタモデルインスタンスとアクションをトリガーしたイベントに関する情報を持つすべての属性と値のペアが含まれます。ペイロードをカスタマイズすることもできます。詳細については、「AWS IoT Events API リファレンス」の「ペイロード」を参照してください。

> **Note**
>
> AWS IoT Events サービスロールにアタッチされたポリシーが アクセス`iot:Publish`許可を付与していることを確認します。詳細については、「の Identity and Access Management AWS IoT Events」を参照してください。

詳細についてはAWS IoT Events API リファレンスのIotTopicPublishActionを参照してください。

AWS IoT Events

IoT Events action

AWS IoT Events アクションを使用すると、データを入力 AWS IoT Events として に送信できます。サポートされているリージョンのリストについては、「Amazon Web Services 全般のリファレンス」の「AWS IoT Events の エンドポイントとクォータ」を参照してください。

AWS IoT Events を使用すると、機器またはデバイスフリートのオペレーションの障害や変更をモニタリングし、そのようなイベントが発生したときにアクションをトリガーできます。詳細については、「 AWS IoT Events デベロッパーガイド」の「 とは AWS IoT Events」を参照してください。

More information (2)

にデータを送信するときは AWS IoT Events、次のパラメータを指定する必要があります。

`inputName`

データを受信する AWS IoT Events 入力の名前。

`payload`

(オプション) デフォルトのペイロードには、ディテクターモデルインスタンスとアクションをトリガーしたイベントに関する情報を持つすべての属性と値のペアが含まれます。ペイロードをカスタマイズすることもできます。詳細については、「AWS IoT Events API リファレンス」の「ペイロード」を参照してください。

> **Note**
>
> AWS IoT Events サービスロールにアタッチされたポリシーが アクセス`iotevents:BatchPutMessage`許可を付与していることを確認します。詳細については、「の Identity and Access Management AWS IoT Events」を参照してください。

詳細については、「AWS IoT Events API リファレンス」の「IotEventsAction」を参照してください。

AWS IoT SiteWise

IoT SiteWise action

AWS IoT SiteWise アクションを使用すると、 のアセットプロパティにデータを送信できます AWS IoT SiteWise。サポートされているリージョンのリストについては、「Amazon Web Services 全般のリファレンス」の「AWS IoT SiteWise の エンドポイントとクォータ」を参照してください。

AWS IoT SiteWise は、産業機器から大規模にデータを収集、整理、分析できるマネージドサービスです。詳細については、「AWS IoT SiteWise ユーザーガイド」の「AWS IoT SiteWiseとは」を参照してください。

More information (11)

のアセットプロパティにデータを送信するときは AWS IoT SiteWise、次のパラメータを指定する必要があります。

> **Important**
>
> データを受信するには、 AWS IoT SiteWiseの既存のアセットプロパティを使用する必要があります。
>
> - AWS IoT Events コンソールを使用する場合は、 を指定`propertyAlias`してターゲットアセットプロパティを識別する必要があります。
> - を使用する場合は AWS CLI、 `assetId`と のいずれか`propertyAlias`または両方を指定`propertyId`して、ターゲットアセットプロパティを識別する必要があります。

詳細については、「AWS IoT SiteWise ユーザーガイド」の「産業用データストリームのアセットプロパティへのマッピング」を参照してください。

propertyAlias

(オプション) アセットプロパティのエイリアス。表現を指定することもできます。

assetId

(オプション) 指定されたプロパティを持つアセットの ID。表現を指定することもできます。

propertyId

(オプション) アセットプロパティの ID。表現を指定することもできます。

entryId

(オプション) このエントリの一意な識別子。エントリ ID を使用すると、障害が発生した場合にエラーの原因となっているデータエントリを追跡できます。デフォルトは、新しい一意の識別子です。表現を指定することもできます。

propertyValue

プロパティ値の詳細を含む構造体。

quality

(オプション) アセットプロパティ値の品質。値は `GOOD`、`BAD`、または `UNCERTAIN` である必要があります。表現を指定することもできます。

timestamp

(オプション) タイムスタンプ情報を含む構造体。この値を指定しない場合、デフォルトはイベント時刻です。

timeInSeconds

Unix エポック形式のタイムスタンプ (秒単位)。有効な範囲は 1 ~ 31556889864403199 です。表現を指定することもできます。

offsetInNanos

(オプション) `timeInSeconds` から変換されたナノ秒オフセット。有効な範囲は 0 ~ 999999999 です。表現を指定することもできます。

value

アセットプロパティ値を含む構造体。

> **Important**
>
> 指定されたアセットプロパティの `dataType` に応じ、値の型として次のいずれかを指定する必要があります。詳細については、「AWS IoT SiteWise API リファレンス」の「AssetProperty」を参照してください。

booleanValue

(オプション) アセットプロパティ値は、ブール値で、`TRUE` または `FALSE` である必要があります。表現を指定することもできます。表現を使用する場合、評価結果はブール値である必要があります。

doubleValue

(オプション) アセットプロパティ値はダブルです。表現を指定することもできます。表現を使用する場合、評価結果はダブルになります。

integerValue

(オプション) アセットプロパティ値は整数です。表現を指定することもできます。表現を使用する場合、評価結果は整数である必要があります。

stringValue

(オプション) アセットプロパティ値は文字列です。表現を指定することもできます。表現を使用する場合は、評価結果が文字列になる必要があります。

> **Note**
>
> AWS IoT Events サービスロールにアタッチされたポリシーが アクセス`iotsitewise:BatchPutAssetPropertyValue`許可を付与していることを確認します。詳細については、「の Identity and Access Management AWS IoT Events」を参照してください。

詳細については、「AWS IoT Events API リファレンス」の「IotSiteWiseAction」を参照してください。

Amazon DynamoDB

DynamoDB action

Amazon DynamoDB アクションを使用して、DynamoDB テーブルにデータを送信することができます。DynamoDB テーブルの 1 つの列は、指定したアクションペイロード内のすべての属性と値のペアを受け取ります。サポートされているリージョンのリストについては、「Amazon Web Services 全般のリファレンス」の「Amazon DynamoDB のエンドポイントとクォータ」を参照してください。

Amazon DynamoDB は、フルマネージド NoSQL データベースサービスであり、シームレスなスケーラビリティを備えた高速で予測可能なパフォーマンスを提供します。詳細については、「Amazon DynamoDB デベロッパーガイド」の「DynamoDBとは」を参照してください。

More information (10)

DynamoDB テーブルの 1 つの列にデータを送信する場合は、次のパラメータを指定する必要があります。

`tableName`

データを受信する DynamoDB テーブルの名前。`tableName` 値は、DynamoDB テーブルのテーブル名と一致する必要があります。表現を指定することもできます。

`hashKeyField`

ハッシュキー (パーティションキーとも呼ばれます) の名前。`hashKeyField` 値は、DynamoDB テーブルのパーティションキーと一致する必要があります。表現を指定することもできます。

`hashKeyType`

(オプション) ハッシュキーのデータ型。ハッシュキータイプの値は `STRING` または `NUMBER` である必要があります。デフォルト: `STRING`。表現を指定することもできます。

`hashKeyValue`

ハッシュキーの値。`hashKeyValue` は、置換テンプレートを使用します。これらのテンプレートは、ランタイム時にデータが提供されます。表現を指定することもできます。

rangeKeyField

(オプション) 範囲キー (ソートキーとも呼ばれます) の名前。rangeKeyField 値は、DynamoDB テーブルのソートキーと一致する必要があります。表現を指定することもできます。

rangeKeyType

(オプション) 範囲キーのデータ型。ハッシュキータイプの値は STRING または NUMBER である必要があります。デフォルト: STRING。表現を指定することもできます。

rangeKeyValue

(オプション) 範囲キーの値。rangeKeyValue は、置換テンプレートを使用します。これらのテンプレートは、ランタイム時にデータが提供されます。表現を指定することもできます。

オペレーション

(オプション) 実行するオペレーションの種類。表現を指定することもできます。オペレーション値は、以下のいずれかの値である必要があります。

- INSERT - データを新しい項目として DynamoDB テーブルに挿入します。これは、デフォルト値です。
- UPDATE - DynamoDB テーブルの既存の項目を新しいデータで更新します。
- DELETE - DynamoDB テーブルから既存の項目を削除します。

payloadField

(オプション) アクションペイロードを受信する DynamoDB の列の名前。デフォルト名は payload です。表現を指定することもできます。

payload

(オプション) デフォルトのペイロードには、ディテクターモデルインスタンスとアクションをトリガーしたイベントに関する情報を持つすべての属性と値のペアが含まれます。ペイロードをカスタマイズすることもできます。詳細については、「AWS IoT Events API リファレンス」の「ペイロード」を参照してください。

指定されたペイロードタイプが文字列の場合、DynamoDBAction は非 JSON データをバイナリデータとして DynamoDB テーブルに送信します。DynamoDB コンソールは、データを Base64-encoded テキストとして表示します。payloadField 値は *payload-field*_raw です。表現を指定することもできます。

> **Note**
>
> AWS IoT Events サービスロールにアタッチされたポリシーが アクセス`dynamodb:PutItem`許可を付与していることを確認します。詳細については、「の Identity and Access Management AWS IoT Events」を参照してください。

詳細については、「AWS IoT Events API リファレンス」の「DynamoDBAction」を参照してください。

Amazon DynamoDB (v2)

DynamoDBv2 action

Amazon DynamoDB (v2) アクションを使用して、DynamoDB テーブルにデータを書き込むことができます。DynamoDB　テーブルの別の列は、指定したアクションペイロードで 1 つの属性と値のペアを受け取ります。サポートされているリージョンのリストについては、「Amazon Web Services 全般のリファレンス」の「Amazon DynamoDB のエンドポイントとクォータ」を参照してください。

Amazon DynamoDB は、フルマネージド NoSQL データベースサービスであり、シームレスなスケーラビリティを備えた高速で予測可能なパフォーマンスを提供します。詳細については、「Amazon DynamoDB デベロッパーガイド」の「DynamoDBとは」を参照してください。

More information (2)

DynamoDB テーブルの複数の列にデータを送信する場合は、次のパラメータを指定する必要があります。

`tableName`

データを受信する DynamoDB テーブルの名前。表現を指定することもできます。

`payload`

(オプション) デフォルトのペイロードには、ディテクターモデルインスタンスとアクションをトリガーしたイベントに関する情報を持つすべての属性と値のペアが含まれます。ペイロードをカスタマイズすることもできます。詳細については、「AWS IoT Events API リファレンス」の「ペイロード」を参照してください。

> **Important**
>
> ペイロードタイプは JSON である必要があります。表現を指定することもできます。

> **Note**
>
> AWS IoT Events サービスロールにアタッチされたポリシーが アクセス`dynamodb:PutItem`許可を付与していることを確認します。詳細については、「の Identity and Access Management AWS IoT Events」を参照してください。

詳細については、「AWS IoT Events API リファレンス」の「DynamoDBv2Action」を参照してください。

Amazon Data Firehose

Firehose action

Amazon Data Firehose アクションを使用すると、Firehose 配信ストリームにデータを送信できます。サポートされているリージョンのリストについては、「」の「Amazon Data Firehose エンドポイントとクォータ」を参照してくださいAmazon Web Services 全般のリファレンス。

Amazon Data Firehose は、Amazon Simple Storage Service (Amazon Simple Storage Service)、Amazon Redshift、Amazon OpenSearch Service (OpenSearch Service)、Splunk などの送信先にリアルタイムのストリーミングデータを配信するためのフルマネージドサービスです。詳細については、Amazon Data Firehose デベロッパーガイドの「Amazon Data Firehose とは」を参照してください。

More information (3)

Firehose 配信ストリームにデータを送信するときは、次のパラメータを指定する必要があります。

`deliveryStreamName`

データを受信する Firehose 配信ストリームの名前。

separator

(オプション) 文字区切り文字を使用して、Firehose 配信ストリームに送信される連続データを分離できます。区切り文字の値は、`'\n'` (改行)、`'\t'` (タブ)、`'\r\n'` (Windows の改行)、または `','` (コンマ) である必要があります。

payload

(オプション) デフォルトのペイロードには、ディテクターモデルインスタンスとアクションをトリガーしたイベントに関する情報を持つすべての属性と値のペアが含まれます。ペイロードをカスタマイズすることもできます。詳細については、「AWS IoT Events API リファレンス」の「ペイロード」を参照してください。

> **Note**
>
> AWS IoT Events サービスロールにアタッチされたポリシーが アクセス`firehose:PutRecord`許可を付与していることを確認します。詳細については、「の Identity and Access Management AWS IoT Events」を参照してください。

詳細については、「AWS IoT Events API リファレンス」の「FirehoseAction」を参照してください。

AWS Lambda

Lambda action

AWS Lambda アクションを使用すると、Lambda 関数を呼び出すことができます。サポートされているリージョンのリストについては、「Amazon Web Services 全般のリファレンス」の「AWS Lambda の エンドポイントとクォータ」を参照してください。

AWS Lambda は、サーバーのプロビジョニングや管理を行わずにコードを実行できるようにするコンピューティングサービスです。詳細については、「 AWS Lambda デベロッパーガイド」の「 とは AWS Lambda」を参照してください。

More information (2)

Lambda 関数を呼び出す際に、以下のパラメータを指定する必要があります。

`functionArn`

呼び出す Lambda 関数の ARN。

`payload`

(オプション) デフォルトのペイロードには、ディテクターモデルインスタンスとアクションをトリガーしたイベントに関する情報を持つすべての属性と値のペアが含まれます。ペイロードをカスタマイズすることもできます。詳細については、「AWS IoT Events API リファレンス」の「ペイロード」を参照してください。

> **Note**
>
> AWS IoT Events サービスロールにアタッチされたポリシーが アクセス`lambda:InvokeFunction`許可を付与していることを確認します。詳細については、「の Identity and Access Management AWS IoT Events」を参照してください。

詳細については、「AWS IoT Events API リファレンス」の「LambdaAction」を参照してください。

Amazon Simple Notification Service

SNS action

Amazon SNS トピックの発行アクションを使用して、Amazon SNS メッセージを発行できます。サポートされているリージョンのリストについては、「Amazon Web Services 全般のリファレンス」の「Amazon Simple Notification Service のエンドポイントとクォータ」を参照してください。

Amazon Simple Notification Service (Amazon Simple Notification Service) は、サブスクライブしているエンドポイントまたはクライアントへのメッセージの配信または送信を調整および管理するウェブサービスです。詳細については、「Amazon Simple Notification Service デベロッパーガイド」の「Amazon SNS とは」を参照してください。

> **Note**
>
> Amazon SNS のトピック発行アクションは、Amazon SNS FIFO (first in, first out) トピックに対応していません。ルールエンジンは完全に分散されたサービスであるた

め、Amazon SNS アクションが開始されたときに、メッセージが指定された順序で表示されない場合があります。

More information (2)

Amazon SNS メッセージを発行するときは、以下のパラメータを指定する必要があります。

`targetArn`

メッセージを受信する Amazon SNS ターゲットの ARN。

`payload`

(オプション) デフォルトのペイロードには、ディテクターモデルインスタンスとアクションをトリガーしたイベントに関する情報を持つすべての属性と値のペアが含まれます。ペイロードをカスタマイズすることもできます。詳細については、「AWS IoT Events API リファレンス」の「ペイロード」を参照してください。

> **Note**
>
> AWS IoT Events サービスロールにアタッチされたポリシーが アクセス`sns:Publish`許可を付与していることを確認します。詳細については、「の Identity and Access Management AWS IoT Events」を参照してください。

詳細については、「AWS IoT Events API リファレンス」の「SNSTopicPublishAction」を参照してください。

Amazon Simple Queue Service

SQS action

Amazon SQS アクションを使用して、Amazon SQS キューにデータを送信することができます。サポートされているリージョンのリストについては、「Amazon Web Services 全般のリファレンス」の「Amazon Simple Queue Service のエンドポイントとクォータ」を参照してください。

Amazon Simple Queue Service (Amazon SQS) は、配信ソフトウェアシステムとコンポーネントを統合および分離できる、安全で耐久性があり、利用可能なホスト型キューを提供します。詳細については、「Amazon Simple Queue Service デベロッパーガイド」の「Amazon Simple Queue Service とは」を参照してください。

> **Note**
>
> Amazon SQS アクションは、>Amazon SQS FIFO (先入れ先出し) トピックをサポートしていません。ルールエンジンは完全に分散されたサービスであるため、Amazon SQS アクションが開始されたときに、メッセージが指定された順序で表示されない場合があります。

More information (3)

Amazon SQS キューにデータを送信するときは、次のパラメータを指定する必要があります。

`queueUrl`

データを受信する Amazon SQS キューの URL。

`useBase64`

(オプション) を指定すると、 はデータを Base64 テキストに AWS IoT Events エンコードしますTRUE。デフォルト: FALSE。

`payload`

(オプション) デフォルトのペイロードには、ディテクターモデルインスタンスとアクションをトリガーしたイベントに関する情報を持つすべての属性と値のペアが含まれます。ペイロードをカスタマイズすることもできます。詳細については、「AWS IoT Events API リファレンス」の「ペイロード」を参照してください。

> **Note**
>
> AWS IoT Events サービスロールにアタッチされたポリシーが アクセスsqs:SendMessage許可を付与していることを確認します。詳細については、「の Identity and Access Management AWS IoT Events」を参照してください。

詳細については、「AWS IoT Events API リファレンス」の「SNSTopicPublishAction」を参照してください。

Amazon SNS と AWS IoT Core ルールエンジンを使用して 関数を AWS Lambda トリガーすることもできます。これにより、Amazon Connect などの他のサービスや、社内のエンタープライズリソースプランニング (ERP) アプリケーションを使用するアクションの実行が可能になります。

> **Note**
>
> データレコードの大きなストリームをリアルタイムで収集して処理するには、Amazon Kinesis などの他の AWS サービスを使用できます。そこから、最初の分析を完了し、結果をディテクターへの入力 AWS IoT Events として に送信できます。

イベントデータをフィルタリング、変換、処理する式

式は、受信データを評価し、計算を実行し、特定のアクションまたは状態遷移が発生する条件を決定するために使用されます。 AWS IoT Events には、ディテクターモデルを作成および更新するときに値を指定するいくつかの方法が用意されています。式を使用してリテラル値を指定するか、特定の値を指定する前に式を評価 AWS IoT Events できます。

トピック

- デバイスデータをフィルタリングし、 でアクションを定義する構文 AWS IoT Events
- の式の例と使用法 AWS IoT Events

デバイスデータをフィルタリングし、 でアクションを定義する構文 AWS IoT Events

式は、デバイスデータをフィルタリングし、アクションを定義するための構文を提供します。 AWS IoT Events 表現では、リテラル、演算子、関数、リファレンス、および置換テンプレートを使用できます。これらのコンポーネントを組み合わせることで、強力で柔軟な式を作成して、IoT データの処理、計算の実行、文字列の操作、ディテクターモデル内での論理的な意思決定を行うことができます。

リテラル

- 整数
- 10 進数
- String
- ブール値

演算子

単項

- 否定 (ブール値): !
- 否定 (ビット単位): ~
- マイナス (算術): -

String

- 連結: **+**

  両方のオペランドは文字列である必要があります。文字列リテラルは一重引用符 (') で囲む必要があります。

  例: `'my' + 'string'` → `'mystring'`

算術

- 加算: **+**

  両方のオペランドは数値である必要があります。

- 減算: **-**
- 除算: **/**

  オペランドの少なくとも 1 つ (除数または被除数) が 10 進値でない限り、除算の結果は丸められた整数値になります。

- 乗算: *****

ビット単位 (整数)

- または: **|**

  例: 13 | 5 → 13

- AND: **&**

  例: 13 & 5 → 5

- XOR: **^**

  例: 13 ^ 5 → 8

- NOT: **~**

  例: ~13 → -14

ブール値

- 未満: **<**
- 以下: **<=**
- 等しい: **==**
- 等しくない: **!=**
- 以上: **>=**

- 以上: **>**
- AND: **&&**
- または: **||**

> **Note**
>
> `||` の部分式に未定義のデータが含まれている場合、その部分式は `false` として扱われます。

括弧

括弧を使用して、表現内の用語をグループ化できます。

AWS IoT Events 式で使用する関数

AWS IoT Events には、ディテクターモデル式の機能を強化するための一連の組み込み関数が用意されています。これらの関数は、タイマー管理、型変換、null チェック、トリガータイプの識別、入力検証、文字列操作、ビット単位オペレーションを有効にします。これらの機能を活用することで、応答性の高い AWS IoT Events 処理ロジックを作成し、IoT アプリケーションの全体的な有効性を向上させることができます。

組み込み関数

`timeout``("`*`timer-name`*`")`

指定されたタイマーが経過した場合、`true` に評価されます。「*`timer-name`*」を、定義したタイマーの名前を引用符で囲んで置き換えます。イベントアクションでは、タイマーを定義してから、タイマーをスタート、リセット、または以前に定義したタイマーをクリアできます。フィールド `detectorModelDefinition.states.onInput|onEnter|onExit.events.actions.setTimer.timerName` を参照してください。

ある状態で設定されたタイマーは、別の状態で参照できます。タイマーが参照されている状態に入力する前に、タイマーを作成した状態にアクセスする必要があります。

たとえば、ディテクターモデルには `TemperatureChecked`と の 2 つの状態があります`RecordUpdated`。`TemperatureChecked` 状態でタイマーを作成しました。`TemperatureChecked` 状態のタイマーを使用するには、まず `RecordUpdated`状態にアクセスする必要があります。

精度を確保するために、タイマーを設定する必要がある最小時間は 60 秒です。

> **Note**
>
> `timeout()` は、実際のタイマーの有効期限切れに最初にチェックされたときにのみ `true` を返し、その後 `false` を返します。

`convert`(*`type`*, *`expression`*)

指定された型に変換された表現の値に評価されます。*###*値は、`String`、`Boolean`、`Decimal` またはである必要があります。これらのキーワードの 1 つ、またはキーワードを含む文字列に評価される表現を使用してください。次の変換のみが成功し、有効な値を返します。

- ブール型 → 文字列

  `"true"` または `"false"` の文字列を返します。
- 10 進数 → 文字列
- 文字列 → ブール型
- 文字列 → 10 進数

  指定する文字列は、10 進数の有効な表現である必要があります。そうでない場合、`convert()` は失敗します。

`convert()` が有効な値を返さない場合、そのパートである表現も無効です。この結果は `false` と同等であり、表現が発生するイベントのパートとして指定された `actions` または `nextState` への移行をトリガーしません。

`isNull`(*`expression`*)

表現がヌルを返す場合、`true` と評価されます。例えば、入力 `{ "a": null }` がメッセージ `true` を受信した場合、以下は `isUndefined($input.MyInput.a)` と評価されますが、`false` は `MyInput` と評価されます。

```
isNull($input.MyInput.a)
```

`isUndefined`(*`expression`*)

表現が未定義の場合、`true` と評価されます。例えば、入力 `MyInput` がメッセージ `{ "a": null }` を受信した場合、以下は `false` と評価されますが、`isNull($input.MyInput.a)` は `true` と評価されます。

```
isUndefined($input.MyInput.a)
```

triggerType("*type*")

###値は "Message" または "Timer" にすることができます。次の例のように、タイマーの期限が切れたために表示されるイベント条件が評価されている場合は、true に評価されます。

```
triggerType("Timer")
```

または、入力メッセージを受信しました。

```
triggerType("Message")
```

currentInput("*input*")

指定された入力メッセージを受信したために表示されるイベント条件が評価されている場合、true に評価されます。例えば、入力 Command がメッセージ { "value": "Abort" } を受信した場合、以下は true と評価されます。

```
currentInput("Command")
```

次の表現のように、この関数を使用して、特定の入力が受信され、タイマーが期限切れになっていないために条件が評価されていることを確認します。

```
currentInput("Command") && $input.Command.value == "Abort"
```

文字列照合関数

startsWith(*expression1*, *expression2*)

最初の文字列表現が 2 番目の文字列表現でスタート場合、true と評価されます。例えば、入力 MyInput がメッセージ { "status": "offline"} を受信した場合、以下は true と評価されます。

```
startsWith($input.MyInput.status, "off")
```

両方の表現は文字列値に評価される必要があります。いずれかの表現が文字列値に評価されない場合、関数の結果は未定義です。変換は実行されません。

endsWith(*expression1*, *expression2*)

最初の文字列表現が 2 番目の文字列表現で終了する場合、true と評価されます。例えば、入力 MyInput がメッセージ { "status": "offline" } を受信した場合、以下は true と評価されます。

```
endsWith($input.MyInput.status, "line")
```

両方の表現は文字列値に評価される必要があります。いずれかの表現が文字列値に評価されない場合、関数の結果は未定義です。変換は実行されません。

contains(*expression1*, *expression2*)

最初の文字列表現に 2 番目の文字列表現が含まれている場合、true と評価されます。例えば、入力 MyInput がメッセージ { "status": "offline" } を受信した場合、以下は true と評価されます。

```
contains($input.MyInput.value, "fli")
```

両方の表現は文字列値に評価される必要があります。いずれかの表現が文字列値に評価されない場合、関数の結果は未定義です。変換は実行されません。

ビット単位の整数操作関数

bitor(*expression1*, *expression2*)

整数表現のビットごとの OR を評価します (バイナリ OR オペレーションは、整数の対応するビットに対して実行されます)。例えば、入力 MyInput がメッセージ { "value1": 13, "value2": 5 } を受信した場合、以下は 13 と評価されます。

```
bitor($input.MyInput.value1, $input.MyInput.value2)
```

両方の表現は整数値に評価される必要があります。いずれかの表現が整数値に評価されない場合、関数の結果は未定義です。変換は実行されません。

bitand(*expression1*, *expression2*)

整数表現のビット単位の AND を評価します (バイナリ AND オペレーションは、整数の対応するビットに対して実行されます)。例えば、入力 MyInput がメッセージ { "value1": 13, "value2": 5 } を受信した場合、以下は 5 と評価されます。

```
bitand($input.MyInput.value1, $input.MyInput.value2)
```

両方の表現は整数値に評価される必要があります。いずれかの表現が整数値に評価されない場合、関数の結果は未定義です。変換は実行されません。

bitxor(*expression1*, *expression2*)

整数表現のビット単位の XOR を評価します (バイナリ XOR オペレーションは、整数の対応するビットに対して実行されます)。例えば、入力 `MyInput` がメッセージ `{ "value1": 13, "value2": 5 }` を受信した場合、以下は 8 と評価されます。

```
bitxor($input.MyInput.value1, $input.MyInput.value2)
```

両方の表現は整数値に評価される必要があります。いずれかの表現が整数値に評価されない場合、関数の結果は未定義です。変換は実行されません。

bitnot(*expression*)

整数表現のビット単位の NOT を評価します (バイナリ NOT オペレーションは整数のビットに対して実行されます)。例えば、入力 `MyInput` がメッセージ `{ "value": 13 }` を受信した場合、以下は `-14` と評価されます。

```
bitnot($input.MyInput.value)
```

両方の表現は整数値に評価される必要があります。いずれかの表現が整数値に評価されない場合、関数の結果は未定義です。変換は実行されません。

AWS IoT Events 式の入力と変数の リファレンス

入力

`$input.`*`input-name`*`.`*`path-to-data`*

`input-name` は、CreateInput アクションを使用して作成する入力です。

例えば、`inputDefinition.attributes.jsonPath` エントリを定義した `TemperatureInput` という名前の入力がある場合、値は次の使用可能なフィールドに表示される可能性があります。

```
{
    "temperature": 78.5,
```

```
    "date": "2018-10-03T16:09:09Z"
  }
```

temperature フィールドの値をリファレンスするには、次のコマンドを使用します。

```
$input.TemperatureInput.temperature
```

値が配列であるフィールドの場合、[*n*] を使用して配列のメンバーをリファレンスできます。例えば、次の値が与えられます。

```
{
    "temperatures": [
      78.4,
      77.9,
      78.8
    ],
    "date": "2018-10-03T16:09:09Z"
  }
```

値 78.8 は、次のコマンドで参照できます。

```
$input.TemperatureInput.temperatures[2]
```

可変

$variable.*variable-name*

variable-name は、CreateDetectorModel アクションを使用して定義した可変です。

例えば、detectorDefinition.states.onInputEvents.actions.setVariable.variableName を使用して定義した TechnicianID という名前の可変がある場合、次のコマンドを使用して、可変に最後に指定された (文字列) 値をリファレンスできます。

```
$variable.TechnicianID
```

可変の値は、setVariable アクションを使用してのみ設定できます。表現の可変に値を割り当てることはできません。可変を設定解除することはできません。例えば、値 null を割り当てることはできません。

> **Note**
>
> (正規表現) パターン `[a-zA-Z][a-zA-Z0-9_]*` に従わない識別子を使用する参照では、それらの識別子をバッククォート (`) で囲む必要があります。例えば、`_value` という名前のフィールドを持つ `MyInput` という名前の入力へのリファレンスでは、このフィールドを `$input.MyInput.`_value`` として指定する必要があります。

表現でリファレンスを使用する場合は、以下をチェックしてください。

- 1 つ以上の演算子でオペランドとしてリファレンスを使用する場合は、参照するすべてのデータ型に互換性があることを確認してください。

  例えば、次の表現では、整数 2 は `==` 演算子と `&&` 演算子の両方のオペランドです。オペランドに互換性があることを確認するには、`$variable.testVariable + 1` と `$variable.testVariable` は整数または 10 進数をリファレンスする必要があります。

  また、整数 1 は `+` 演算子のオペランドです。したがって、`$variable.testVariable` は整数または 10 進数をリファレンスする必要があります。

  ```
  '$variable.testVariable + 1 == 2 && $variable.testVariable'
  ```

- 関数に渡される引数としてリファレンスを使用する場合は、関数が参照するデータ型をサポートしていることを確認してください。

  例えば、次の `timeout("time-name")` 関数では、引数として二重引用符を含む文字列が必要です。*`timer-name`* 値のリファレンスを使用する場合は、二重引用符で囲まれた文字列を参照する必要があります。

  ```
  timeout("timer-name")
  ```

  > **Note**
  >
  > `convert(type, expression)` 関数の場合、*`###`*値にリファレンスを使用する場合、リファレンスの評価結果は `String`、`Decimal`、または `Boolean` である必要があります。

AWS IoT Events 式は、整数、10 進数、文字列、ブール値データ型をサポートしています。次の表に、互換性のないタイプのペアのリストを示します。

互換性のないタイプのペア
整数、文字列
整数、ブール型
10 進数、文字列
10 進数、ブール型
文字列、ブール型

AWS IoT Events 式の置換テンプレート

```
'${expression}'
```

${} は、文字列を補間された文字列として識別します。は任意の AWS IoT Events 式にexpressionすることができます。これには、演算子、関数、およびリファレンスが含まれます。

例えば、SetVariableAction アクションを使用して可変を定義しました。variableName は SensorID であり、value は 10 です。次の置換テンプレートを作成できます。

置換テンプレート	結果文字列
`'${'Sensor ' + $variable.SensorID}'`	"Sensor 10"
`'Sensor ' + '${$variable.SensorID + 1}'`	"Sensor 11"
`'Sensor 10: ${$variable.SensorID == 10}'`	"Sensor 10: true"

置換テンプレート	結果文字列
`'{\"sensor\":\"${$variable.SensorID + 1}\"}'`	`"{\"sensor"\:\"11\"}"`
`'{\"sensor\":${$variable.SensorID + 1}}'`	`"{\"sensor\":11}"`

の式の例と使用法 AWS IoT Events

次の方法で、ディテクターモデルの値を指定できます。

- AWS IoT Events コンソールでサポートされている式を入力します。
- 式をパラメータとして AWS IoT Events APIs に渡します。

表現は、リテラル、演算子、関数、参照、および置換テンプレートをサポートします。

> **Important**
> 表現は、整数、10 進数、文字列、またはブール値をリファレンスする必要があります。

AWS IoT Events 式の記述

AWS IoT Events 式を記述するには、次の例を参照してください。

リテラル

リテラル値の場合、表現には一重引用符が含まれている必要があります。ブール値は `true` または `false` のいずれかでなければなりません。

```
'123'        # Integer
'123.12'     # Decimal
'hello'      # String
'true'       # Boolean
```

リファレンス

リファレンス用に、可変または入力値のいずれかを指定する必要があります。

- 次の入力は、10 進数 `10.01` を参照しています。

```
$input.GreenhouseInput.temperature
```

- 次の可変は、文字列 `Greenhouse Temperature Table` を参照します。

```
$variable.TableName
```

置換テンプレート

置換テンプレートの場合、`${}` を使用する必要があり、テンプレートは一重引用符で囲む必要があります。置換テンプレートには、リテラル、演算子、関数、リファレンス、および置換テンプレートの組み合わせを含めることもできます。

- 次の表現の評価結果は文字列 `50.018 in Fahrenheit` です。

```
'${$input.GreenhouseInput.temperature * 9 / 5 + 32} in Fahrenheit'
```

- 次の表現の評価結果は文字列 `{\"sensor_id\":\"Sensor_1\",\"temperature\":\"50.018\"}` です。

```
'{\"sensor_id\":\"${$input.GreenhouseInput.sensors[0].sensor1}\",\"temperature\":
\"${$input.GreenhouseInput.temperature*9/5+32}\"}'
```

文字列の連結

文字列の連結には、`+` を使用する必要があります。文字列の連結には、リテラル、演算子、関数、リファレンス、および置換テンプレートの組み合わせを含めることもできます。

- 次の表現の評価結果は文字列 `Greenhouse Temperature Table 2000-01-01` です。

```
'Greenhouse Temperature Table ' + $input.GreenhouseInput.date
```

AWS IoT Events ディテクターモデルの例

このページでは、さまざまな AWS IoT Events 機能の設定方法を示すユースケースの例を一覧表示します。例としては、温度しきい値などの基本的な検出から、より高度な異常検出や機械学習シナリオまで多岐にわたります。各例には、 AWS IoT Events 検出、アクション、統合の設定に役立つ手順とコードスニペットが含まれています。これらの例は、 AWS IoT Events サービスの柔軟性と、さまざまな IoT アプリケーションやユースケースに合わせてカスタマイズする方法を示しています。 AWS IoT Events 機能を調べるとき、または特定の検出または自動化ワークフローを実装するガイダンスが必要な場合は、このページを参照してください。

トピック

- 例: での HIPAA 温度制御の使用 AWS IoT Events
- 例: を使用したクレーン検出条件 AWS IoT Events
- で検出された条件に応じてコマンドを送信する AWS IoT Events
- クレーンモニタリング用の AWS IoT Events ディテクターモデル
- AWS IoT Events クレーンモニタリング用の 入力
- でアラームと運用メッセージを送信する AWS IoT Events
- 例: センサーとアプリケーションによる AWS IoT Events イベント検出
- 例: Device HeartBeat で とのデバイス接続をモニタリングする AWS IoT Events
- 例: の ISA アラーム AWS IoT Events
- 例: を使用してシンプルなアラームを構築する AWS IoT Events

例: での HIPAA 温度制御の使用 AWS IoT Events

背景

この例では、次の特徴を備えた温度制御モデル (サーモスタット) を実装します。

- 複数の領域をモニタリングおよび制御できる、定義した 1 つのディテクターモデル。(ディテクーインスタンスはエリアごとに作成されます。)
- 各ディテクーインスタンスは、各制御領域に配置された複数のセンサーから温度データを受信します。
- 各エリアの希望温度 (設定温度) はいつでも変更できます。
- 各エリアの操作パラメータを定義し、これらのパラメータをいつでも変更できます。

- エリアにセンサーを追加したり、エリアからセンサーを削除したりすることはいつでもできます。
- 単位を損傷から保護するために、時間加熱および冷却単位の最小実行を有効にすることができます。
- ディテクターは、異常なセンサー読み取り値を拒否して報告します。
- 緊急時の温度設定値を定義できます。いずれかのセンサーが定義した設定値より上または下の温度を報告すると、加熱または冷却単位がすぐに作動し、ディテクターがその温度スパイクを報告します。

この例は、次の機能を示しています。

- イベントディテクターモデルを作成します。
- 入力を作成します。
- ディテクターモデルに入力を取り込みます。
- トリガー条件を評価します。
- 条件の状態可変を参照し、条件に応じて可変の値を設定します。
- 条件内のタイマーを参照し、条件に応じてタイマーを設定します。
- Amazon SNS および MQTT メッセージを送信するアクションを実行します。

の PVC システムの入力定義 AWS IoT Events

`seedTemperatureInput` は、エリアのディテクターインスタンスを作成し、その操作パラメータを定義するために使用されます。

で PVC システムの入力を設定する AWS IoT Events ことは、効果的な気候制御にとって重要です。この例では、温度、湿度、稼働率、エネルギー消費データなどのパラメータをキャプチャする入力を設定する方法を示します。入力属性の定義、データソースの設定、ディテクターモデルが最適な管理と効率のために正確でタイムリーな情報を受け取るのに役立つ前処理ルールの設定について説明します。

使用した CLI コマンド:

```
aws iotevents create-input --cli-input-json file://seedInput.json
```

ファイル: `seedInput.json`

```
{
  "inputName": "seedTemperatureInput",
  "inputDescription": "Temperature seed values.",
  "inputDefinition": {
    "attributes": [
      { "jsonPath": "areaId" },
      { "jsonPath": "desiredTemperature" },
      { "jsonPath": "allowedError" },
      { "jsonPath": "rangeHigh" },
      { "jsonPath": "rangeLow" },
      { "jsonPath": "anomalousHigh" },
      { "jsonPath": "anomalousLow" },
      { "jsonPath": "sensorCount" },
      { "jsonPath": "noDelay" }
    ]
  }
}
```

レスポンス:

```
{
    "inputConfiguration": {
        "status": "ACTIVE",
        "inputArn": "arn:aws:iotevents:us-west-2:123456789012:input/
seedTemperatureInput",
        "lastUpdateTime": 1557519620.736,
        "creationTime": 1557519620.736,
        "inputName": "seedTemperatureInput",
        "inputDescription": "Temperature seed values."
    }
}
```

`temperatureInput` は、必要に応じて、各エリアの各センサーから送信される必要があります。

使用した CLI コマンド:

```
aws iotevents create-input --cli-input-json file://temperatureInput.json
```

ファイル: `temperatureInput.json`

```
{
```

```
  "inputName": "temperatureInput",
  "inputDescription": "Temperature sensor unit data.",
  "inputDefinition": {
    "attributes": [
      { "jsonPath": "sensorId" },
      { "jsonPath": "areaId" },
      { "jsonPath": "sensorData.temperature" }
    ]
  }
}
```

レスポンス:

```
{
    "inputConfiguration": {
        "status": "ACTIVE",
        "inputArn": "arn:aws:iotevents:us-west-2:123456789012:input/temperatureInput",
        "lastUpdateTime": 1557519707.399,
        "creationTime": 1557519707.399,
        "inputName": "temperatureInput",
        "inputDescription": "Temperature sensor unit data."
    }
}
```

を使用した PVC システムのディテクターモデル定義 AWS IoT Events

`areaDetectorModel` は、ディテクターインスタンスがどのように機能するかを定義します。各 `state machine` インスタンスは、温度センサーの読み取り値を取り込み、状態を変更して、これらの読み取り値に応じて制御メッセージを送信します。

使用した CLI コマンド:

```
aws iotevents create-detector-model --cli-input-json file://areaDetectorModel.json
```

ファイル: `areaDetectorModel.json`

```
{
  "detectorModelName": "areaDetectorModel",
  "detectorModelDefinition": {
    "states": [
      {
```

```
        "stateName": "start",
        "onEnter": {
          "events": [
            {
              "eventName": "prepare",
              "condition": "true",
              "actions": [
                {
                  "setVariable": {
                    "variableName": "sensorId",
                    "value": "0"
                  }
                },
                {
                  "setVariable": {
                    "variableName": "reportedTemperature",
                    "value": "0.1"
                  }
                },
                {
                  "setVariable": {
                    "variableName": "resetMe",
                    "value": "false"
                  }
                }
              ]
            }
          ]
        },
        "onInput": {
          "transitionEvents": [
            {
              "eventName": "initialize",
              "condition": "$input.seedTemperatureInput.sensorCount > 0",
              "actions": [
                {
                  "setVariable": {
                    "variableName": "rangeHigh",
                    "value": "$input.seedTemperatureInput.rangeHigh"
                  }
                },
                {
                  "setVariable": {
                    "variableName": "rangeLow",
```

```
                "value": "$input.seedTemperatureInput.rangeLow"
              }
            },
            {
              "setVariable": {
                "variableName": "desiredTemperature",
                "value": "$input.seedTemperatureInput.desiredTemperature"
              }
            },
            {
              "setVariable": {
                "variableName": "averageTemperature",
                "value": "$input.seedTemperatureInput.desiredTemperature"
              }
            },
            {
              "setVariable": {
                "variableName": "allowedError",
                "value": "$input.seedTemperatureInput.allowedError"
              }
            },
            {
              "setVariable": {
                "variableName": "anomalousHigh",
                "value": "$input.seedTemperatureInput.anomalousHigh"
              }
            },
            {
              "setVariable": {
                "variableName": "anomalousLow",
                "value": "$input.seedTemperatureInput.anomalousLow"
              }
            },
            {
              "setVariable": {
                "variableName": "sensorCount",
                "value": "$input.seedTemperatureInput.sensorCount"
              }
            },
            {
              "setVariable": {
                "variableName": "noDelay",
                "value": "$input.seedTemperatureInput.noDelay == true"
              }
```

```
                }
              ],
              "nextState": "idle"
            },
            {
              "eventName": "reset",
              "condition": "($variable.resetMe == true) &&
($input.temperatureInput.sensorData.temperature < $variable.anomalousHigh &&
$input.temperatureInput.sensorData.temperature > $variable.anomalousLow)",
              "actions": [
                {
                  "setVariable": {
                    "variableName": "averageTemperature",
                    "value": "((($variable.averageTemperature * ($variable.sensorCount
- 1)) + $input.temperatureInput.sensorData.temperature) / $variable.sensorCount)"
                  }
                }
              ],
              "nextState": "idle"
            }
          ]
        },
        "onExit": {
          "events": [
            {
              "eventName": "resetHeatCool",
              "condition": "true",
              "actions": [
                {
                  "sns": {
                    "targetArn": "arn:aws:sns:us-west-2:123456789012:heatOff"
                  }
                },
                {
                  "sns": {
                    "targetArn": "arn:aws:sns:us-west-2:123456789012:coolOff"
                  }
                },
                {
                  "iotTopicPublish": {
                    "mqttTopic": "hvac/Heating/Off"
                  }
                },
                {
```

```
                "iotTopicPublish": {
                  "mqttTopic": "hvac/Cooling/Off"
                }
              }
            ]
          }
        ]
      }
    },


    {
      "stateName": "idle",
      "onInput": {
        "events": [
          {
            "eventName": "whatWasInput",
            "condition": "true",
            "actions": [
              {
                "setVariable": {
                  "variableName": "sensorId",
                  "value": "$input.temperatureInput.sensorId"
                }
              },
              {
                "setVariable": {
                  "variableName": "reportedTemperature",
                  "value": "$input.temperatureInput.sensorData.temperature"
                }
              }
            ]
          },
          {
            "eventName": "changeDesired",
            "condition": "$input.seedTemperatureInput.desiredTemperature !=
$variable.desiredTemperature",
            "actions": [
              {
                "setVariable": {
                  "variableName": "desiredTemperature",
                  "value": "$input.seedTemperatureInput.desiredTemperature"
                }
              }
```

```
              ]
            },
            {
              "eventName": "calculateAverage",
              "condition": "$input.temperatureInput.sensorData.temperature <
$variable.anomalousHigh && $input.temperatureInput.sensorData.temperature >
$variable.anomalousLow",
              "actions": [
                {
                  "setVariable": {
                    "variableName": "averageTemperature",
                    "value": "((($variable.averageTemperature * ($variable.sensorCount
- 1)) + $input.temperatureInput.sensorData.temperature) / $variable.sensorCount)"
                  }
                }
              ]
            }
          ],
          "transitionEvents": [
            {
              "eventName": "anomalousInputArrived",
              "condition": "$input.temperatureInput.sensorData.temperature >=
$variable.anomalousHigh || $input.temperatureInput.sensorData.temperature <=
$variable.anomalousLow",
              "actions": [
                {
                  "iotTopicPublish": {
                    "mqttTopic": "temperatureSensor/anomaly"
                  }
                }
              ],
              "nextState": "idle"
            },

            {
              "eventName": "highTemperatureSpike",
              "condition": "$input.temperatureInput.sensorData.temperature >
$variable.rangeHigh",
              "actions": [
                {
                  "iotTopicPublish": {
                    "mqttTopic": "temperatureSensor/spike"
                  }
                },
```

```
                {
                  "sns": {
                    "targetArn": "arn:aws:sns:us-west-2:123456789012:coolOn"
                  }
                },
                {
                  "iotTopicPublish": {
                    "mqttTopic": "hvac/Cooling/On"
                  }
                },
                {
                  "setVariable": {
                    "variableName": "enteringNewState",
                    "value": "true"
                  }
                }
              ],
              "nextState": "cooling"
            },

            {
              "eventName": "lowTemperatureSpike",
              "condition": "$input.temperatureInput.sensorData.temperature <
$variable.rangeLow",
              "actions": [
                {
                  "iotTopicPublish": {
                    "mqttTopic": "temperatureSensor/spike"
                  }
                },
                {
                  "sns": {
                    "targetArn": "arn:aws:sns:us-west-2:123456789012:heatOn"
                  }
                },
                {
                  "iotTopicPublish": {
                    "mqttTopic": "hvac/Heating/On"
                  }
                },
                {
                  "setVariable": {
                    "variableName": "enteringNewState",
                    "value": "true"
```

```
                }
              }
            ],
            "nextState": "heating"
          },

          {
            "eventName": "highTemperatureThreshold",
            "condition": "((((($variable.averageTemperature * ($variable.sensorCount
- 1)) + $input.temperatureInput.sensorData.temperature) / $variable.sensorCount) >
($variable.desiredTemperature + $variable.allowedError))",
            "actions": [
              {
                "sns": {
                  "targetArn": "arn:aws:sns:us-west-2:123456789012:coolOn"
                }
              },
              {
                "iotTopicPublish": {
                  "mqttTopic": "hvac/Cooling/On"
                }
              },
              {
                "setVariable": {
                  "variableName": "enteringNewState",
                  "value": "true"
                }
              }
            ],
            "nextState": "cooling"
          },

          {
            "eventName": "lowTemperatureThreshold",
            "condition": "((((($variable.averageTemperature * ($variable.sensorCount
- 1)) + $input.temperatureInput.sensorData.temperature) / $variable.sensorCount) <
($variable.desiredTemperature - $variable.allowedError))",
            "actions": [
              {
                "sns": {
                  "targetArn": "arn:aws:sns:us-west-2:123456789012:heatOn"
                }
              },
              {
```

```
                "iotTopicPublish": {
                  "mqttTopic": "hvac/Heating/On"
                }
              },
              {
                "setVariable": {
                  "variableName": "enteringNewState",
                  "value": "true"
                }
              }
            ],
            "nextState": "heating"
          }
        ]
      }
    },


    {
      "stateName": "cooling",
      "onEnter": {
        "events": [
          {
            "eventName": "delay",
            "condition": "!$variable.noDelay && $variable.enteringNewState",
            "actions": [
              {
                "setTimer": {
                  "timerName": "coolingTimer",
                  "seconds": 180
                }
              },
              {
                "setVariable": {
                  "variableName": "goodToGo",
                  "value": "false"
                }
              }
            ]
          },
          {
            "eventName": "dontDelay",
            "condition": "$variable.noDelay == true",
            "actions": [
```

```
                {
                  "setVariable": {
                    "variableName": "goodToGo",
                    "value": "true"
                  }
                }
              ]
            },
            {
              "eventName": "beenHere",
              "condition": "true",
              "actions": [
                {
                  "setVariable": {
                    "variableName": "enteringNewState",
                    "value": "false"
                  }
                }
              ]
            }
          ]
        },

        "onInput": {
          "events": [
            {
              "eventName": "whatWasInput",
              "condition": "true",
              "actions": [
                {
                  "setVariable": {
                    "variableName": "sensorId",
                    "value": "$input.temperatureInput.sensorId"
                  }
                },
                {
                  "setVariable": {
                    "variableName": "reportedTemperature",
                    "value": "$input.temperatureInput.sensorData.temperature"
                  }
                }
              ]
            },
            {
```

```
              "eventName": "changeDesired",
              "condition": "$input.seedTemperatureInput.desiredTemperature !=
$variable.desiredTemperature",
              "actions": [
                {
                  "setVariable": {
                    "variableName": "desiredTemperature",
                    "value": "$input.seedTemperatureInput.desiredTemperature"
                  }
                }
              ]
            },
            {
              "eventName": "calculateAverage",
              "condition": "$input.temperatureInput.sensorData.temperature <
$variable.anomalousHigh && $input.temperatureInput.sensorData.temperature >
$variable.anomalousLow",
              "actions": [
                {
                  "setVariable": {
                    "variableName": "averageTemperature",
                    "value": "((($variable.averageTemperature * ($variable.sensorCount
- 1)) + $input.temperatureInput.sensorData.temperature) / $variable.sensorCount)"
                  }
                }
              ]
            },
            {
              "eventName": "areWeThereYet",
              "condition": "(timeout(\"coolingTimer\"))",
              "actions": [
                {
                  "setVariable": {
                    "variableName": "goodToGo",
                    "value": "true"
                  }
                }
              ]
            }
          ],
          "transitionEvents": [
            {
              "eventName": "anomalousInputArrived",
```

```
              "condition": "$input.temperatureInput.sensorData.temperature >=
$variable.anomalousHigh || $input.temperatureInput.sensorData.temperature <=
$variable.anomalousLow",
              "actions": [
                {
                  "iotTopicPublish": {
                    "mqttTopic": "temperatureSensor/anomaly"
                  }
                }
              ],
              "nextState": "cooling"
            },

            {
              "eventName": "highTemperatureSpike",
              "condition": "$input.temperatureInput.sensorData.temperature >
$variable.rangeHigh",
              "actions": [
                {
                  "iotTopicPublish": {
                    "mqttTopic": "temperatureSensor/spike"
                  }
                }
              ],
              "nextState": "cooling"
            },

            {
              "eventName": "lowTemperatureSpike",
              "condition": "$input.temperatureInput.sensorData.temperature <
$variable.rangeLow",
              "actions": [
                {
                  "iotTopicPublish": {
                    "mqttTopic": "temperatureSensor/spike"
                  }
                },
                {
                  "sns": {
                    "targetArn": "arn:aws:sns:us-west-2:123456789012:coolOff"
                  }
                },
                {
                  "sns": {
```

```
                "targetArn": "arn:aws:sns:us-west-2:123456789012:heatOn"
              }
            },
            {
              "iotTopicPublish": {
                "mqttTopic": "hvac/Cooling/Off"
              }
            },
            {
              "iotTopicPublish": {
                "mqttTopic": "hvac/Heating/On"
              }
            },
            {
              "setVariable": {
                "variableName": "enteringNewState",
                "value": "true"
              }
            }
          ],
          "nextState": "heating"
        },

        {
          "eventName": "desiredTemperature",
          "condition": "((((($variable.averageTemperature * ($variable.sensorCount
- 1)) + $input.temperatureInput.sensorData.temperature) / $variable.sensorCount) <=
($variable.desiredTemperature - $variable.allowedError)) && $variable.goodToGo ==
true",
          "actions": [
            {
              "sns": {
                "targetArn": "arn:aws:sns:us-west-2:123456789012:coolOff"
              }
            },
            {
              "iotTopicPublish": {
                "mqttTopic": "hvac/Cooling/Off"
              }
            }
          ],
          "nextState": "idle"
        }
      ]
```

```
        }
      },


      {
        "stateName": "heating",
        "onEnter": {
          "events": [
            {
              "eventName": "delay",
              "condition": "!$variable.noDelay && $variable.enteringNewState",
              "actions": [
                {
                  "setTimer": {
                    "timerName": "heatingTimer",
                    "seconds": 120
                  }
                },
                {
                  "setVariable": {
                    "variableName": "goodToGo",
                    "value": "false"
                  }
                }
              ]
            },
            {
              "eventName": "dontDelay",
              "condition": "$variable.noDelay == true",
              "actions": [
                {
                  "setVariable": {
                    "variableName": "goodToGo",
                    "value": "true"
                  }
                }
              ]
            },
            {
              "eventName": "beenHere",
              "condition": "true",
              "actions": [
                {
                  "setVariable": {
```

```
                    "variableName": "enteringNewState",
                    "value": "false"
                  }
                }
              ]
            }
          ]
        },

        "onInput": {
          "events": [
            {
              "eventName": "whatWasInput",
              "condition": "true",
              "actions": [
                {
                  "setVariable": {
                    "variableName": "sensorId",
                    "value": "$input.temperatureInput.sensorId"
                  }
                },
                {
                  "setVariable": {
                    "variableName": "reportedTemperature",
                    "value": "$input.temperatureInput.sensorData.temperature"
                  }
                }
              ]
            },
            {
              "eventName": "changeDesired",
              "condition": "$input.seedTemperatureInput.desiredTemperature !=
$variable.desiredTemperature",
              "actions": [
                {
                  "setVariable": {
                    "variableName": "desiredTemperature",
                    "value": "$input.seedTemperatureInput.desiredTemperature"
                  }
                }
              ]
            },
            {
              "eventName": "calculateAverage",
```

```
              "condition": "$input.temperatureInput.sensorData.temperature <
$variable.anomalousHigh && $input.temperatureInput.sensorData.temperature >
$variable.anomalousLow",
              "actions": [
                {
                  "setVariable": {
                    "variableName": "averageTemperature",
                    "value": "((($variable.averageTemperature * ($variable.sensorCount
- 1)) + $input.temperatureInput.sensorData.temperature) / $variable.sensorCount)"
                  }
                }
              ]
            },
            {
              "eventName": "areWeThereYet",
              "condition": "(timeout(\"heatingTimer\"))",
              "actions": [
                {
                  "setVariable": {
                    "variableName": "goodToGo",
                    "value": "true"
                  }
                }
              ]
            }
          ],
          "transitionEvents": [
            {
              "eventName": "anomalousInputArrived",
              "condition": "$input.temperatureInput.sensorData.temperature >=
$variable.anomalousHigh || $input.temperatureInput.sensorData.temperature <=
$variable.anomalousLow",
              "actions": [
                {
                  "iotTopicPublish": {
                    "mqttTopic": "temperatureSensor/anomaly"
                  }
                }
              ],
              "nextState": "heating"
            },

            {
              "eventName": "highTemperatureSpike",
```

```
              "condition": "$input.temperatureInput.sensorData.temperature >
$variable.rangeHigh",
              "actions": [
                {
                  "iotTopicPublish": {
                    "mqttTopic": "temperatureSensor/spike"
                  }
                },
                {
                  "sns": {
                    "targetArn": "arn:aws:sns:us-west-2:123456789012:heatOff"
                  }
                },
                {
                  "sns": {
                    "targetArn": "arn:aws:sns:us-west-2:123456789012:coolOn"
                  }
                },
                {
                  "iotTopicPublish": {
                    "mqttTopic": "hvac/Heating/Off"
                  }
                },
                {
                  "iotTopicPublish": {
                    "mqttTopic": "hvac/Cooling/On"
                  }
                },
                {
                  "setVariable": {
                    "variableName": "enteringNewState",
                    "value": "true"
                  }
                }
              ],
              "nextState": "cooling"
            },

            {
              "eventName": "lowTemperatureSpike",
              "condition": "$input.temperatureInput.sensorData.temperature <
$variable.rangeLow",
              "actions": [
                {
```

```
                  "iotTopicPublish": {
                    "mqttTopic": "temperatureSensor/spike"
                  }
                }
              ],
              "nextState": "heating"
            },

            {
              "eventName": "desiredTemperature",
              "condition": "((((($variable.averageTemperature * ($variable.sensorCount
 - 1)) + $input.temperatureInput.sensorData.temperature) / $variable.sensorCount) >=
 ($variable.desiredTemperature + $variable.allowedError)) && $variable.goodToGo ==
 true",
              "actions": [
                {
                  "sns": {
                    "targetArn": "arn:aws:sns:us-west-2:123456789012:heatOff"
                  }
                },
                {
                  "iotTopicPublish": {
                    "mqttTopic": "hvac/Heating/Off"
                  }
                }
              ],
              "nextState": "idle"
            }
          ]
        }
      }

    ],

    "initialStateName": "start"
  },
  "key": "areaId",
  "roleArn": "arn:aws:iam::123456789012:role/IoTEventsRole"
}
```

レスポンス:

```
{
```

```
    "detectorModelConfiguration": {
        "status": "ACTIVATING",
        "lastUpdateTime": 1557523491.168,
        "roleArn": "arn:aws:iam::123456789012:role/IoTEventsRole",
        "creationTime": 1557523491.168,
        "detectorModelArn": "arn:aws:iotevents:us-west-2:123456789012:detectorModel/
areaDetectorModel",
        "key": "areaId",
        "detectorModelName": "areaDetectorModel",
        "detectorModelVersion": "1"
    }
}
```

の PVC システムの BatchPutMessage の例 AWS IoT Events

この例では、BatchPutMessage を使用して、エリアのディテクターインスタンスを作成し、初期動作パラメータを定義します。

使用した CLI コマンド:

```
aws iotevents-data batch-put-message --cli-input-json file://seedExample.json --cli-
binary-format raw-in-base64-out
```

ファイル: seedExample.json

```
{
  "messages": [
    {
      "messageId": "00001",
      "inputName": "seedTemperatureInput",
      "payload": "{\"areaId\": \"Area51\", \"desiredTemperature\": 20.0, \"allowedError
\": 0.7, \"rangeHigh\": 30.0, \"rangeLow\": 15.0, \"anomalousHigh\": 60.0,
 \"anomalousLow\": 0.0, \"sensorCount\": 10, \"noDelay\": false}"
    }
  ]
}
```

レスポンス:

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

この例では、BatchPutMessage を使用して、エリア内の 1 つのセンサーの温度センサーの読み取り値を報告します。

使用した CLI コマンド:

```
aws iotevents-data batch-put-message --cli-input-json file://temperatureExample.json --
cli-binary-format raw-in-base64-out
```

ファイル: temperatureExample.json

```
{
  "messages": [
    {
      "messageId": "00005",
      "inputName": "temperatureInput",
      "payload": "{\"sensorId\": \"05\", \"areaId\": \"Area51\", \"sensorData\":
 {\"temperature\": 23.12} }"
    }
  ]
}
```

レスポンス:

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

この例では、BatchPutMessage を使用してエリアの目的の温度を変更します。

使用した CLI コマンド:

```
aws iotevents-data batch-put-message --cli-input-json file://seedSetDesiredTemp.json --
cli-binary-format raw-in-base64-out
```

ファイル: seedSetDesiredTemp.json

```
{
  "messages": [
    {
      "messageId": "00001",
      "inputName": "seedTemperatureInput",
      "payload": "{\"areaId\": \"Area51\", \"desiredTemperature\": 23.0}"
```

```
    }
  ]
}
```

レスポンス:

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

Area51 ディテクターインスタンスによって生成された Amazon SNS メッセージの例:

```
Heating system off command> {
  "eventTime":1557520274729,
  "payload":{
    "actionExecutionId":"f3159081-bac3-38a4-96f7-74af0940d0a4",
    "detector":{
      "detectorModelName":"areaDetectorModel",
      "keyValue":"Area51",
      "detectorModelVersion":"1"
    },
    "eventTriggerDetails":{
      "inputName":"seedTemperatureInput",
      "messageId":"00001",
      "triggerType":"Message"
    },
    "state":{
      "stateName":"start",
      "variables":{
        "sensorCount":10,
        "rangeHigh":30.0,
        "resetMe":false,
        "enteringNewState":true,
        "averageTemperature":20.0,
        "rangeLow":15.0,
        "noDelay":false,
        "allowedError":0.7,
        "desiredTemperature":20.0,
        "anomalousHigh":60.0,
        "reportedTemperature":0.1,
        "anomalousLow":0.0,
```

```
        "sensorId":0
      },
      "timers":{}
    }
  },
  "eventName":"resetHeatCool"
}
```

```
Cooling system off command> {
  "eventTime":1557520274729,
  "payload":{
    "actionExecutionId":"98f6a1b5-8f40-3cdb-9256-93afd4d66192",
    "detector":{
      "detectorModelName":"areaDetectorModel",
      "keyValue":"Area51",
      "detectorModelVersion":"1"
    },
    "eventTriggerDetails":{
      "inputName":"seedTemperatureInput",
      "messageId":"00001",
      "triggerType":"Message"
    },
    "state":{
      "stateName":"start",
      "variables":{
        "sensorCount":10,
        "rangeHigh":30.0,
        "resetMe":false,
        "enteringNewState":true,
        "averageTemperature":20.0,
        "rangeLow":15.0,
        "noDelay":false,
        "allowedError":0.7,
        "desiredTemperature":20.0,
        "anomalousHigh":60.0,
        "reportedTemperature":0.1,
        "anomalousLow":0.0,
        "sensorId":0
      },
      "timers":{}
    }
  },
```

```
  "eventName":"resetHeatCool"
}
```

この例では、DescribeDetector API を使用して、ディテクターインスタンスの現在の状態に関する情報を取得します。

```
aws iotevents-data describe-detector --detector-model-name areaDetectorModel --key-
value Area51
```

レスポンス:

```
{
    "detector": {
        "lastUpdateTime": 1557521572.216,
        "creationTime": 1557520274.405,
        "state": {
            "variables": [
                {
                    "name": "resetMe",
                    "value": "false"
                },
                {
                    "name": "rangeLow",
                    "value": "15.0"
                },
                {
                    "name": "noDelay",
                    "value": "false"
                },
                {
                    "name": "desiredTemperature",
                    "value": "20.0"
                },
                {
                    "name": "anomalousLow",
                    "value": "0.0"
                },
                {
                    "name": "sensorId",
                    "value": "\"01\""
                },
                {
                    "name": "sensorCount",
```

```
                    "value": "10"
                },
                {
                    "name": "rangeHigh",
                    "value": "30.0"
                },
                {
                    "name": "enteringNewState",
                    "value": "false"
                },
                {
                    "name": "averageTemperature",
                    "value": "19.572"
                },
                {
                    "name": "allowedError",
                    "value": "0.7"
                },
                {
                    "name": "anomalousHigh",
                    "value": "60.0"
                },
                {
                    "name": "reportedTemperature",
                    "value": "15.72"
                },
                {
                    "name": "goodToGo",
                    "value": "false"
                }
            ],
            "stateName": "idle",
            "timers": [
                {
                    "timestamp": 1557520454.0,
                    "name": "idleTimer"
                }
            ]
        },
        "keyValue": "Area51",
        "detectorModelName": "areaDetectorModel",
        "detectorModelVersion": "1"
    }
```

```
}
```

の PVC システムの BatchUpdateDetector の例 AWS IoT Events

この例では、`BatchUpdateDetector` を使用して、動作中のディテクターインスタンスの操作パラメータを変更します。

効率的な Hadoop システム管理では、複数のディテクターのバッチ更新が必要になることがよくあります。このセクションでは、ディテクターに AWS IoT Eventsのバッチ更新機能を使用する方法を示します。複数のコントロールパラメータを同時に変更し、しきい値を更新して、デバイスのフリート全体でレスポンスアクションを調整し、大規模なシステムを効果的に管理する能力を向上させる方法について説明します。

使用した CLI コマンド:

```
aws iotevents-data batch-update-detector --cli-input-json file://areaDM.BUD.json
```

ファイル: `areaDM.BUD.json`

```
{
  "detectors": [
    {
      "messageId": "0001",
      "detectorModelName": "areaDetectorModel",
      "keyValue": "Area51",
      "state": {
        "stateName": "start",
        "variables": [
          {
            "name": "desiredTemperature",
            "value": "22"
          },
          {
            "name": "averageTemperature",
            "value": "22"
          },
          {
            "name": "allowedError",
            "value": "1.0"
          },
          {
```

```
            "name": "rangeHigh",
            "value": "30.0"
          },
          {
            "name": "rangeLow",
            "value": "15.0"
          },
          {
            "name": "anomalousHigh",
            "value": "60.0"
          },
          {
            "name": "anomalousLow",
            "value": "0.0"
          },
          {
            "name": "sensorCount",
            "value": "12"
          },
          {
            "name": "noDelay",
            "value": "true"
          },
          {
            "name": "goodToGo",
            "value": "true"
          },
          {
            "name": "sensorId",
            "value": "0"
          },
          {
            "name": "reportedTemperature",
            "value": "0.1"
          },
          {
            "name": "resetMe",
            "value": "true"
          }
        ],
        "timers": [
        ]
      }
    }
```

```
  ]
}
```

レスポンス:

```
{
    An error occurred (InvalidRequestException) when calling the BatchUpdateDetector
 operation: Number of variables in the detector exceeds the limit 10
}
```

AWS IoT Core ルールエンジンと AWS IoT Events

次のルールは、 AWS IoT Events MQTT メッセージをシャドウ更新リクエストメッセージとして再発行します。 AWS IoT Core ディテクターモデルによって制御される各エリアの加熱単位と冷却単位に対してモノが定義されていることを前提としています。

この例では、`Area51HeatingUnit` および `Area51CoolingUnit` という名前のものを定義しました。

使用した CLI コマンド:

```
aws iot create-topic-rule --cli-input-json file://ADMShadowCoolOffRule.json
```

ファイル: `ADMShadowCoolOffRule.json`

```
{
  "ruleName": "ADMShadowCoolOff",
  "topicRulePayload": {
    "sql": "SELECT topic(3) as state.desired.command FROM 'hvac/Cooling/Off'",
    "description": "areaDetectorModel mqtt topic publish to cooling unit shadow
 request",
    "ruleDisabled": false,
    "awsIotSqlVersion": "2016-03-23",
    "actions": [
      {
        "republish": {
          "topic": "$$aws/things/${payload.detector.keyValue}CoolingUnit/shadow/
update",
          "roleArn": "arn:aws:iam::123456789012:role/service-role/ADMShadowRole"
        }
      }
```

```
    ]
  }
}
```

レスポンス: [空]

使用した CLI コマンド:

```
aws iot create-topic-rule --cli-input-json file://ADMShadowCoolOnRule.json
```

ファイル: `ADMShadowCoolOnRule.json`

```
{
  "ruleName": "ADMShadowCoolOn",
  "topicRulePayload": {
    "sql": "SELECT topic(3) as state.desired.command FROM 'hvac/Cooling/On'",
    "description": "areaDetectorModel mqtt topic publish to cooling unit shadow
 request",
    "ruleDisabled": false,
    "awsIotSqlVersion": "2016-03-23",
    "actions": [
      {
        "republish": {
          "topic": "$$aws/things/${payload.detector.keyValue}CoolingUnit/shadow/
update",
          "roleArn": "arn:aws:iam::123456789012:role/service-role/ADMShadowRole"
        }
      }
    ]
  }
}
```

レスポンス: [空]

使用した CLI コマンド:

```
aws iot create-topic-rule --cli-input-json file://ADMShadowHeatOffRule.json
```

ファイル: `ADMShadowHeatOffRule.json`

```
{
```

```
  "ruleName": "ADMShadowHeatOff",
  "topicRulePayload": {
    "sql": "SELECT topic(3) as state.desired.command FROM 'hvac/Heating/Off'",
    "description": "areaDetectorModel mqtt topic publish to heating unit shadow
 request",
    "ruleDisabled": false,
    "awsIotSqlVersion": "2016-03-23",
    "actions": [
      {
        "republish": {
          "topic": "$$aws/things/${payload.detector.keyValue}HeatingUnit/shadow/
update",
          "roleArn": "arn:aws:iam::123456789012:role/service-role/ADMShadowRole"
        }
      }
    ]
  }
}
```

レスポンス: [空]

使用した CLI コマンド:

```
aws iot create-topic-rule --cli-input-json file://ADMShadowHeatOnRule.json
```

ファイル: `ADMShadowHeatOnRule.json`

```
{
  "ruleName": "ADMShadowHeatOn",
  "topicRulePayload": {
    "sql": "SELECT topic(3) as state.desired.command FROM 'hvac/Heating/On'",
    "description": "areaDetectorModel mqtt topic publish to heating unit shadow
 request",
    "ruleDisabled": false,
    "awsIotSqlVersion": "2016-03-23",
    "actions": [
      {
        "republish": {
          "topic": "$$aws/things/${payload.detector.keyValue}HeatingUnit/shadow/
update",
          "roleArn": "arn:aws:iam::123456789012:role/service-role/ADMShadowRole"
        }
      }
```

```
    ]
  }
}
```

レスポンス: [空]

例: を使用したクレーン検出条件 AWS IoT Events

多くのクレーンのオペレーターは、機械のメンテナンスまたは交換が必要な時期を検出し、適切な通知をトリガーしたいと考えています。各クレーンにはモーターがあります。モーターは、圧力と温度に関する情報を含むメッセージ (入力) を送信します。オペレーターは、2 つのレベルのイベントディテクターを必要としています。

- クレーンレベルのイベントディテクター
- モーターレベルのイベントディテクター

モーターからのメッセージ (`craneId` と `motorid` の両方のメタデータを含む) を使用して、オペレーターは適切なルーティングを使用して両方のレベルのイベントディテクターを実行できます。イベント条件が満たされた場合、通知は適切な Amazon SNS トピックに送信される必要があります。オペレーターは、重複する通知が発生しないようにディテクターモデルを設定できます。

この例は、次の機能を示しています。

- 入力の生成、読み取り、更新、削除 (CRUD)。
- イベントディテクターモデルとさまざまなバージョンのイベントディテクターの作成、読み取り、更新、削除 (CRUD)。
- 1 つの入力を複数のイベントディテクターにルーティングします。
- ディテクターモデルへの入力の取り込み。
- トリガー条件とライフサイクルイベントの評価。
- 条件内の状態可変を参照し、条件に応じて可変の値を設定する機能。
- 定義、状態、トリガーエバリュエーター、およびアクションエグゼキューターを使用したランタイムオーケストレーション。
- SNSターゲットを使用した `ActionsExecutor` でのアクションの実行。

で検出された条件に応じてコマンドを送信する AWS IoT Events

このページでは、 AWS IoT Events コマンドを使用して入力の設定、ディテクターモデルの作成、シミュレートされたセンサーデータの送信を行う例を示します。この例では、 AWS IoT Events を活用してモーターやクレーンなどの産業機器を監視する方法を示しています。

```
#Create Pressure Input
aws iotevents create-input  --cli-input-json file://pressureInput.json
aws iotevents describe-input --input-name PressureInput
aws iotevents update-input  --cli-input-json file://pressureInput.json
aws iotevents list-inputs
aws iotevents delete-input --input-name PressureInput

#Create Temperature Input
aws iotevents create-input  --cli-input-json file://temperatureInput.json
aws iotevents describe-input --input-name TemperatureInput
aws iotevents update-input  --cli-input-json file://temperatureInput.json
aws iotevents list-inputs
aws iotevents delete-input --input-name TemperatureInput

#Create Motor Event Detector using pressure and temperature input
aws iotevents create-detector-model  --cli-input-json file://motorDetectorModel.json
aws iotevents describe-detector-model --detector-model-name motorDetectorModel
aws iotevents update-detector-model  --cli-input-json file://
updateMotorDetectorModel.json
aws iotevents list-detector-models
aws iotevents list-detector-model-versions --detector-model-name motorDetectorModel
aws iotevents delete-detector-model --detector-model-name motorDetectorModel

#Create Crane Event Detector using temperature input
aws iotevents create-detector-model  --cli-input-json file://craneDetectorModel.json
aws iotevents describe-detector-model --detector-model-name craneDetectorModel
aws iotevents update-detector-model  --cli-input-json file://
updateCraneDetectorModel.json
aws iotevents list-detector-models
aws iotevents list-detector-model-versions --detector-model-name craneDetectorModel
aws iotevents delete-detector-model --detector-model-name craneDetectorModel

#Replace craneIds
sed -i '' "s/100008/100009/g" messages/*

#Replace motorIds
```

```
sed -i '' "s/200008/200009/g" messages/*

#Send HighPressure message
aws iotevents-data batch-put-message --cli-input-json file://messages/
highPressureMessage.json --cli-binary-format raw-in-base64-out

#Send HighTemperature message
aws iotevents-data batch-put-message --cli-input-json file://messages/
highTemperatureMessage.json --cli-binary-format raw-in-base64-out

#Send LowPressure message
aws iotevents-data batch-put-message --cli-input-json file://messages/
lowPressureMessage.json --cli-binary-format raw-in-base64-out

#Send LowTemperature message
aws iotevents-data batch-put-message --cli-input-json file://messages/
lowTemperatureMessage.json --cli-binary-format raw-in-base64-out
```

クレーンモニタリング用の AWS IoT Events ディテクターモデル

機器またはデバイスフリートのオペレーションの障害や変更をモニタリングし、そのようなイベントが発生したときにアクションをトリガーします。状態、ルール、アクションを指定するディテクターモデルを JSON で定義します。これにより、温度や圧力などの入力のモニタリング、しきい値違反の追跡、アラートの送信を行うことができます。この例では、クレーンとモーターのディテクターモデル、過熱の問題の検出、しきい値を超えたときの Amazon SNS による通知を示しています。モニタリングを中断することなく、モデルを更新して動作を絞り込むことができます。

ファイル: `craneDetectorModel.json`

```
{
    "detectorModelName": "craneDetectorModel",
    "detectorModelDefinition": {
        "states": [
            {
                "stateName": "Running",
                "onEnter": {
                    "events": [
                        {
                            "eventName": "init",
                            "condition": "true",
                            "actions": [
```

```
                                {
                                    "setVariable": {
                                        "variableName": "craneThresholdBreached",
                                        "value": "0"
                                    }
                                }
                            ]
                        }
                    ]
                },
                "onInput": {
                    "events": [
                        {
                            "eventName": "Overheated",
                            "condition": "$input.TemperatureInput.temperature > 35",
                            "actions": [
                                {
                                    "setVariable": {
                                        "variableName": "craneThresholdBreached",
                                        "value": "$variable.craneThresholdBreached + 1"
                                    }
                                }
                            ]
                        },
                        {
                            "eventName": "Crane Threshold Breached",
                            "condition": "$variable.craneThresholdBreached > 5",
                            "actions": [
                                {
                                    "sns": {
                                        "targetArn": "arn:aws:sns:us-
east-1:123456789012:CraneSNSTopic"
                                    }
                                }
                            ]
                        },
                        {
                            "eventName": "Underheated",
                            "condition": "$input.TemperatureInput.temperature < 25",
                            "actions": [
                                {
                                    "setVariable": {
                                        "variableName": "craneThresholdBreached",
                                        "value": "0"
```

```
                                    }
                                }
                            ]
                        }
                    ]
                }
            }
        ],
        "initialStateName": "Running"
    },
    "key": "craneid",
    "roleArn": "arn:aws:iam::123456789012:role/columboSNSRole"
}
```

既存のディテクターモデルを更新する。ファイル: `updateCraneDetectorModel.json`

```
{
    "detectorModelName": "craneDetectorModel",
    "detectorModelDefinition": {
        "states": [
            {
                "stateName": "Running",
                "onEnter": {
                    "events": [
                        {
                            "eventName": "init",
                            "condition": "true",
                            "actions": [
                                {
                                    "setVariable": {
                                        "variableName": "craneThresholdBreached",
                                        "value": "0"
                                    }
                                },
                                {
                                    "setVariable": {
                                        "variableName": "alarmRaised",
                                        "value": "'false'"
                                    }
                                }
                            ]
                        }
                    ]
```

```
                },
                "onInput": {
                    "events": [
                        {
                            "eventName": "Overheated",
                            "condition": "$input.TemperatureInput.temperature > 30",
                            "actions": [
                                {
                                    "setVariable": {
                                        "variableName": "craneThresholdBreached",
                                        "value": "$variable.craneThresholdBreached + 1"
                                    }
                                }
                            ]
                        },
                        {
                            "eventName": "Crane Threshold Breached",
                            "condition": "$variable.craneThresholdBreached > 5 &&
 $variable.alarmRaised == 'false'",
                            "actions": [
                                {
                                    "sns": {
                                        "targetArn": "arn:aws:sns:us-
east-1:123456789012:CraneSNSTopic"
                                    }
                                },
                                {
                                    "setVariable": {
                                        "variableName": "alarmRaised",
                                        "value": "'true'"
                                    }
                                }
                            ]
                        },
                        {
                            "eventName": "Underheated",
                            "condition": "$input.TemperatureInput.temperature < 10",
                            "actions": [
                                {
                                    "setVariable": {
                                        "variableName": "craneThresholdBreached",
                                        "value": "0"
                                    }
                                }
```

```
                            ]
                        }
                    ]
                }
            }
        ],
        "initialStateName": "Running"
    },
    "roleArn": "arn:aws:iam::123456789012:role/columboSNSRole"
}
```

ファイル: motorDetectorModel.json

```
{
    "detectorModelName": "motorDetectorModel",
    "detectorModelDefinition": {
        "states": [
            {
                "stateName": "Running",
                "onEnter": {
                    "events": [
                        {
                            "eventName": "init",
                            "condition": "true",
                            "actions": [
                                {
                                    "setVariable": {
                                        "variableName": "motorThresholdBreached",
                                        "value": "0"
                                    }
                                }
                            ]
                        }
                    ]
                },
                "onInput": {
                    "events": [
                        {
                            "eventName": "Overheated And Overpressurized",
                            "condition": "$input.PressureInput.pressure > 70 &&
 $input.TemperatureInput.temperature > 30",
                            "actions": [
                                {
```

```
                                    "setVariable": {
                                        "variableName": "motorThresholdBreached",
                                        "value": "$variable.motorThresholdBreached + 1"
                                    }
                                }
                            ]
                        },
                        {
                            "eventName": "Motor Threshold Breached",
                            "condition": "$variable.motorThresholdBreached > 5",
                            "actions": [
                                {
                                    "sns": {
                                        "targetArn": "arn:aws:sns:us-
east-1:123456789012:MotorSNSTopic"
                                    }
                                }
                            ]
                        }
                    ]
                }
            }
        ],
        "initialStateName": "Running"
    },
    "key": "motorId",
    "roleArn": "arn:aws:iam::123456789012:role/columboSNSRole"
}
```

既存のディテクターモデルを更新する。ファイル: `updateMotorDetectorModel.json`

```
{
    "detectorModelName": "motorDetectorModel",
    "detectorModelDefinition": {
        "states": [
            {
                "stateName": "Running",
                "onEnter": {
                    "events": [
                        {
                            "eventName": "init",
                            "condition": "true",
                            "actions": [
```

```
                                {
                                    "setVariable": {
                                        "variableName": "motorThresholdBreached",
                                        "value": "0"
                                    }
                                }
                            ]
                        }
                    ]
                },
                "onInput": {
                    "events": [
                        {
                            "eventName": "Overheated And Overpressurized",
                            "condition": "$input.PressureInput.pressure > 70 &&
 $input.TemperatureInput.temperature > 30",
                            "actions": [
                                {
                                    "setVariable": {
                                        "variableName": "motorThresholdBreached",
                                        "value": "$variable.motorThresholdBreached + 1"
                                    }
                                }
                            ]
                        },
                        {
                            "eventName": "Motor Threshold Breached",
                            "condition": "$variable.motorThresholdBreached > 5",
                            "actions": [
                                {
                                    "sns": {
                                        "targetArn": "arn:aws:sns:us-
east-1:123456789012:MotorSNSTopic"
                                    }
                                }
                            ]
                        }
                    ]
                }
            }
        ],
        "initialStateName": "Running"
    },
    "roleArn": "arn:aws:iam::123456789012:role/columboSNSRole"
```

```
}
```

AWS IoT Events クレーンモニタリング用の 入力

この例では、 を使用してクレーンモニタリングシステムの入力を設定する方法を示します AWS IoT Events。圧力と温度の入力をキャプチャして、複雑な産業機器モニタリングの入力を構築する方法を示します。

ファイル: `pressureInput.json`

```
{
    "inputName": "PressureInput",
    "inputDescription": "this is a pressure input description",
    "inputDefinition": {
        "attributes": [
          {"jsonPath": "pressure"}
        ]
    }
}
```

ファイル: `temperatureInput.json`

```
{
    "inputName": "TemperatureInput",
    "inputDescription": "this is temperature input description",
    "inputDefinition": {
        "attributes": [
            {"jsonPath": "temperature"}
        ]
    }
}
```

でアラームと運用メッセージを送信する AWS IoT Events

効果的なメッセージ処理は、クレーンモニタリングシステムで重要です。このセクションでは、クレーンセンサーからのさまざまなメッセージタイプを処理して応答 AWS IoT Events するように を設定する方法を示します。特定のメッセージに基づいてアラームを設定すると、ステータスの更新を解析、フィルタリング、ルーティングして適切なアクションをトリガーするのに役立ちます。

ファイル: `highPressureMessage.json`

```
{
   "messages": [
        {
           "messageId": "1",
           "inputName": "PressureInput",
           "payload": "{\"craneid\": \"100009\", \"pressure\": 80, \"motorid\":
 \"200009\"}"

        }
    ]
}
```

ファイル: highTemperatureMessage.json

```
{
   "messages": [
        {
           "messageId": "2",
           "inputName": "TemperatureInput",
           "payload": "{\"craneid\": \"100009\", \"temperature\": 40, \"motorid\":
 \"200009\"}"
        }
    ]
}
```

ファイル: lowPressureMessage.json

```
{
   "messages": [
        {
           "messageId": "1",
           "inputName": "PressureInput",
           "payload": "{\"craneid\": \"100009\", \"pressure\": 20, \"motorid\":
 \"200009\"}"
        }
    ]
}
```

ファイル: lowTemperatureMessage.json

```
{
   "messages": [
```

```
        {
           "messageId": "2",
           "inputName": "TemperatureInput",
           "payload": "{\"craneid\": \"100009\", \"temperature\": 20, \"motorid\":
 \"200009\"}"
        }
    ]
}
```

例: センサーとアプリケーションによる AWS IoT Events イベント検出

このディテクターモデルは、 AWS IoT Events コンソールから利用可能なテンプレートの 1 つです。便宜上、ここに再掲します。

この例では AWS IoT Events、センサーデータを使用したアプリケーションイベント検出を示します。これは、適切なアクションをトリガーできるように、指定されたイベントをモニタリングするディテクターモデルを作成する方法を示しています。複数のセンサー入力を作成し、複雑なイベント条件を定義し、段階的な応答メカニズムを設定できます。

```
{
    "detectorModelName": "EventDetectionSensorsAndApplications",
    "detectorModelDefinition": {
        "states": [
            {
                "onInput": {
                    "transitionEvents": [],
                    "events": []
                },
                "stateName": "Device_exception",
                "onEnter": {
                    "events": [
                        {
                            "eventName": "Send_mqtt",
                            "actions": [
                                {
                                    "iotTopicPublish": {
                                        "mqttTopic": "Device_stolen"
                                    }
                                }
                            ],
```

```
                            "condition": "true"
                        }
                    ]
                },
                "onExit": {
                    "events": []
                }
            },
            {
                "onInput": {
                    "transitionEvents": [
                        {
                            "eventName": "To_in_use",
                            "actions": [],
                            "condition": "$variable.position !=
 $input.AWS_IoTEvents_Blueprints_Tracking_DeviceInput.gps_position",
                            "nextState": "Device_in_use"
                        }
                    ],
                    "events": []
                },
                "stateName": "Device_idle",
                "onEnter": {
                    "events": [
                        {
                            "eventName": "Set_position",
                            "actions": [
                                {
                                    "setVariable": {
                                        "variableName": "position",
                                        "value":
 "$input.AWS_IoTEvents_Blueprints_Tracking_DeviceInput.gps_position"
                                    }
                                }
                            ],
                            "condition": "true"
                        }
                    ]
                },
                "onExit": {
                    "events": []
                }
            },
            {
```

```
                "onInput": {
                    "transitionEvents": [
                        {
                            "eventName": "To_exception",
                            "actions": [],
                            "condition":
 "$input.AWS_IoTEvents_Blueprints_Tracking_UserInput.device_id !=
 $input.AWS_IoTEvents_Blueprints_Tracking_DeviceInput.device_id",
                            "nextState": "Device_exception"
                        }
                    ],
                    "events": []
                },
                "stateName": "Device_in_use",
                "onEnter": {
                    "events": []
                },
                "onExit": {
                    "events": []
                }
            }
        ],
        "initialStateName": "Device_idle"
    }
}
```

例: Device HeartBeat で とのデバイス接続をモニタリングする AWS IoT Events

このディテクターモデルは、 AWS IoT Events コンソールから利用可能なテンプレートの 1 つです。便宜上、ここに再掲します。

Defective Heart Beat (DHB) の例は、 AWS IoT Events を医療モニタリングで使用する方法を示しています。この例では、心拍数データを分析し、不規則なパターンを検出し、適切なレスポンスをトリガーするディテクターモデルを作成する方法を示します。入力の設定、しきい値の定義、潜在的なリスクの問題に対するアラートの設定について学び、関連する医療アプリケーションで AWS IoT Eventsの汎用性を示します。

```
{
    "detectorModelDefinition": {
        "states": [
```

```
            {
                "onInput": {
                    "transitionEvents": [
                        {
                            "eventName": "To_normal",
                            "actions": [],
                            "condition":
 "currentInput(\"AWS_IoTEvents_Blueprints_Heartbeat_Input\")",
                            "nextState": "Normal"
                        }
                    ],
                    "events": []
                },
                "stateName": "Offline",
                "onEnter": {
                    "events": [
                        {
                            "eventName": "Send_notification",
                            "actions": [
                                {
                                    "sns": {
                                        "targetArn": "sns-topic-arn"
                                    }
                                }
                            ],
                            "condition": "true"
                        }
                    ]
                },
                "onExit": {
                    "events": []
                }
            },
            {
                "onInput": {
                    "transitionEvents": [
                        {
                            "eventName": "Go_offline",
                            "actions": [],
                            "condition": "timeout(\"awake\")",
                            "nextState": "Offline"
                        }
                    ],
                    "events": [
```

```
                        {
                            "eventName": "Reset_timer",
                            "actions": [
                                {
                                    "resetTimer": {
                                        "timerName": "awake"
                                    }
                                }
                            ],
                            "condition":
 "currentInput(\"AWS_IoTEvents_Blueprints_Heartbeat_Input\")"
                        }
                    ]
                },
                "stateName": "Normal",
                "onEnter": {
                    "events": [
                        {
                            "eventName": "Create_timer",
                            "actions": [
                                {
                                    "setTimer": {
                                        "seconds": 300,
                                        "timerName": "awake"
                                    }
                                }
                            ],
                            "condition":
 "$input.AWS_IoTEvents_Blueprints_Heartbeat_Input.value > 0"
                        }
                    ]
                },
                "onExit": {
                    "events": []
                }
            }
        ],
        "initialStateName": "Normal"
    }
}
```

例: の ISA アラーム AWS IoT Events

このディテクターモデルは、 AWS IoT Events コンソールから利用可能なテンプレートの 1 つです。便宜上、ここに再掲します。

```
{
    "detectorModelName": "AWS_IoTEvents_Blueprints_ISA_Alarm",
    "detectorModelDefinition": {
        "states": [
            {
                "onInput": {
                    "transitionEvents": [
                        {
                            "eventName": "unshelve",
                            "actions": [],
                            "condition":
 "$input.AWS_IoTEvents_Blueprints_ISA_Alarm_Input.command == \"unshelve\" &&
 $variable.state == \"rtnunack\"",
                            "nextState": "RTN_Unacknowledged"
                        },
                        {
                            "eventName": "unshelve",
                            "actions": [],
                            "condition":
 "$input.AWS_IoTEvents_Blueprints_ISA_Alarm_Input.command == \"unshelve\" &&
 $variable.state == \"ack\"",
                            "nextState": "Acknowledged"
                        },
                        {
                            "eventName": "unshelve",
                            "actions": [],
                            "condition":
 "$input.AWS_IoTEvents_Blueprints_ISA_Alarm_Input.command == \"unshelve\" &&
 $variable.state == \"unack\"",
                            "nextState": "Unacknowledged"
                        },
                        {
                            "eventName": "unshelve",
                            "actions": [],
                            "condition":
 "$input.AWS_IoTEvents_Blueprints_ISA_Alarm_Input.command == \"unshelve\" &&
 $variable.state == \"normal\"",
                            "nextState": "Normal"
```

```
                        }
                    ],
                    "events": []
                },
                "stateName": "Shelved",
                "onEnter": {
                    "events": []
                },
                "onExit": {
                    "events": []
                }
            },
            {
                "onInput": {
                    "transitionEvents": [
                        {
                            "eventName": "abnormal_condition",
                            "actions": [],
                            "condition":
"$input.AWS_IoTEvents_Blueprints_ISA_Alarm_Input.value > $variable.higher_threshold ||
$input.AWS_IoTEvents_Blueprints_ISA_Alarm_Input.value < $variable.lower_threshold",
                            "nextState": "Unacknowledged"
                        },
                        {
                            "eventName": "acknowledge",
                            "actions": [],
                            "condition":
"$input.AWS_IoTEvents_Blueprints_ISA_Alarm_Input.command == \"acknowledge\"",
                            "nextState": "Normal"
                        },
                        {
                            "eventName": "shelve",
                            "actions": [],
                            "condition":
"$input.AWS_IoTEvents_Blueprints_ISA_Alarm_Input.command == \"shelve\"",
                            "nextState": "Shelved"
                        },
                        {
                            "eventName": "remove_from_service",
                            "actions": [],
                            "condition":
"$input.AWS_IoTEvents_Blueprints_ISA_Alarm_Input.command == \"remove\"",
                            "nextState": "Out_of_service"
                        },
```

```
                        {
                            "eventName": "suppression",
                            "actions": [],
                            "condition":
"$input.AWS_IoTEvents_Blueprints_ISA_Alarm_Input.command == \"suppressed\"",
                            "nextState": "Suppressed_by_design"
                        }
                    ],
                    "events": []
                },
                "stateName": "RTN_Unacknowledged",
                "onEnter": {
                    "events": [
                        {
                            "eventName": "State Save",
                            "actions": [
                                {
                                    "setVariable": {
                                        "variableName": "state",
                                        "value": "\"rtnunack\""
                                    }
                                }
                            ],
                            "condition": "true"
                        }
                    ]
                },
                "onExit": {
                    "events": []
                }
            },
            {
                "onInput": {
                    "transitionEvents": [
                        {
                            "eventName": "abnormal_condition",
                            "actions": [],
                            "condition":
"$input.AWS_IoTEvents_Blueprints_ISA_Alarm_Input.value > $variable.higher_threshold ||
$input.AWS_IoTEvents_Blueprints_ISA_Alarm_Input.value < $variable.lower_threshold",
                            "nextState": "Unacknowledged"
                        },
                        {
                            "eventName": "shelve",
```

```
                            "actions": [],
                            "condition":
"$input.AWS_IoTEvents_Blueprints_ISA_Alarm_Input.command == \"shelve\"",
                            "nextState": "Shelved"
                        },
                        {
                            "eventName": "remove_from_service",
                            "actions": [],
                            "condition":
"$input.AWS_IoTEvents_Blueprints_ISA_Alarm_Input.command == \"remove\"",
                            "nextState": "Out_of_service"
                        },
                        {
                            "eventName": "suppression",
                            "actions": [],
                            "condition":
"$input.AWS_IoTEvents_Blueprints_ISA_Alarm_Input.command == \"suppressed\"",
                            "nextState": "Suppressed_by_design"
                        }
                    ],
                    "events": [
                        {
                            "eventName": "Create Config variables",
                            "actions": [
                                {
                                    "setVariable": {
                                        "variableName": "lower_threshold",
                                        "value":
"$input.AWS_IoTEvents_Blueprints_ISA_Alarm_Input.lower_threshold"
                                    }
                                },
                                {
                                    "setVariable": {
                                        "variableName": "higher_threshold",
                                        "value":
"$input.AWS_IoTEvents_Blueprints_ISA_Alarm_Input.higher_threshold"
                                    }
                                }
                            ],
                            "condition": "$variable.lower_threshold !=
$variable.lower_threshold"
                        }
                    ]
                },
```

```
                "stateName": "Normal",
                "onEnter": {
                    "events": [
                        {
                            "eventName": "State Save",
                            "actions": [
                                {
                                    "setVariable": {
                                        "variableName": "state",
                                        "value": "\"normal\""
                                    }
                                }
                            ],
                            "condition": "true"
                        }
                    ]
                },
                "onExit": {
                    "events": []
                }
            },
            {
                "onInput": {
                    "transitionEvents": [
                        {
                            "eventName": "acknowledge",
                            "actions": [],
                            "condition":
"$input.AWS_IoTEvents_Blueprints_ISA_Alarm_Input.command == \"acknowledge\"",
                            "nextState": "Acknowledged"
                        },
                        {
                            "eventName": "return_to_normal",
                            "actions": [],
                            "condition":
"($input.AWS_IoTEvents_Blueprints_ISA_Alarm_Input.value <= $variable.higher_threshold
&& $input.AWS_IoTEvents_Blueprints_ISA_Alarm_Input.value >=
$variable.lower_threshold)",
                            "nextState": "RTN_Unacknowledged"
                        },
                        {
                            "eventName": "shelve",
                            "actions": [],
```

```
                            "condition":
"$input.AWS_IoTEvents_Blueprints_ISA_Alarm_Input.command == \"shelve\"",
                            "nextState": "Shelved"
                        },
                        {
                            "eventName": "remove_from_service",
                            "actions": [],
                            "condition":
"$input.AWS_IoTEvents_Blueprints_ISA_Alarm_Input.command == \"remove\"",
                            "nextState": "Out_of_service"
                        },
                        {
                            "eventName": "suppression",
                            "actions": [],
                            "condition":
"$input.AWS_IoTEvents_Blueprints_ISA_Alarm_Input.command == \"suppressed\"",
                            "nextState": "Suppressed_by_design"
                        }
                    ],
                    "events": []
                },
                "stateName": "Unacknowledged",
                "onEnter": {
                    "events": [
                        {
                            "eventName": "State Save",
                            "actions": [
                                {
                                    "setVariable": {
                                        "variableName": "state",
                                        "value": "\"unack\""
                                    }
                                }
                            ],
                            "condition": "true"
                        }
                    ]
                },
                "onExit": {
                    "events": []
                }
            },
            {
                "onInput": {
```

```
                    "transitionEvents": [
                        {
                            "eventName": "unsuppression",
                            "actions": [],
                            "condition":
"$input.AWS_IoTEvents_Blueprints_ISA_Alarm_Input.command == \"unsuppressed\" &&
$variable.state == \"normal\"",
                            "nextState": "Normal"
                        },
                        {
                            "eventName": "unsuppression",
                            "actions": [],
                            "condition":
"$input.AWS_IoTEvents_Blueprints_ISA_Alarm_Input.command == \"unsuppressed\" &&
$variable.state == \"unack\"",
                            "nextState": "Unacknowledged"
                        },
                        {
                            "eventName": "unsuppression",
                            "actions": [],
                            "condition":
"$input.AWS_IoTEvents_Blueprints_ISA_Alarm_Input.command == \"unsuppressed\" &&
$variable.state == \"ack\"",
                            "nextState": "Acknowledged"
                        },
                        {
                            "eventName": "unsuppression",
                            "actions": [],
                            "condition":
"$input.AWS_IoTEvents_Blueprints_ISA_Alarm_Input.command == \"unsuppressed\" &&
$variable.state == \"rtnunack\"",
                            "nextState": "RTN_Unacknowledged"
                        }
                    ],
                    "events": []
                },
                "stateName": "Suppressed_by_design",
                "onEnter": {
                    "events": []
                },
                "onExit": {
                    "events": []
                }
            },
```

```
            {
                "onInput": {
                    "transitionEvents": [
                        {
                            "eventName": "return_to_service",
                            "actions": [],
                            "condition":
"$input.AWS_IoTEvents_Blueprints_ISA_Alarm_Input.command == \"add\" && $variable.state
== \"rtnunack\"",
                            "nextState": "RTN_Unacknowledged"
                        },
                        {
                            "eventName": "return_to_service",
                            "actions": [],
                            "condition":
"$input.AWS_IoTEvents_Blueprints_ISA_Alarm_Input.command == \"add\" && $variable.state
== \"unack\"",
                            "nextState": "Unacknowledged"
                        },
                        {
                            "eventName": "return_to_service",
                            "actions": [],
                            "condition":
"$input.AWS_IoTEvents_Blueprints_ISA_Alarm_Input.command == \"add\" && $variable.state
== \"ack\"",
                            "nextState": "Acknowledged"
                        },
                        {
                            "eventName": "return_to_service",
                            "actions": [],
                            "condition":
"$input.AWS_IoTEvents_Blueprints_ISA_Alarm_Input.command == \"add\" && $variable.state
== \"normal\"",
                            "nextState": "Normal"
                        }
                    ],
                    "events": []
                },
                "stateName": "Out_of_service",
                "onEnter": {
                    "events": []
                },
                "onExit": {
                    "events": []
```

```
                }
            },
            {
                "onInput": {
                    "transitionEvents": [
                        {
                            "eventName": "re-alarm",
                            "actions": [],
                            "condition": "timeout(\"snooze\")",
                            "nextState": "Unacknowledged"
                        },
                        {
                            "eventName": "return_to_normal",
                            "actions": [],
                            "condition":
"$input.AWS_IoTEvents_Blueprints_ISA_Alarm_Input.command == \"reset\"",
                            "nextState": "Normal"
                        },
                        {
                            "eventName": "shelve",
                            "actions": [],
                            "condition":
"$input.AWS_IoTEvents_Blueprints_ISA_Alarm_Input.command == \"shelve\"",
                            "nextState": "Shelved"
                        },
                        {
                            "eventName": "remove_from_service",
                            "actions": [],
                            "condition":
"$input.AWS_IoTEvents_Blueprints_ISA_Alarm_Input.command == \"remove\"",
                            "nextState": "Out_of_service"
                        },
                        {
                            "eventName": "suppression",
                            "actions": [],
                            "condition":
"$input.AWS_IoTEvents_Blueprints_ISA_Alarm_Input.command == \"suppressed\"",
                            "nextState": "Suppressed_by_design"
                        }
                    ],
                    "events": []
                },
                "stateName": "Acknowledged",
                "onEnter": {
```

```
                    "events": [
                        {
                            "eventName": "Create Timer",
                            "actions": [
                                {
                                    "setTimer": {
                                        "seconds": 60,
                                        "timerName": "snooze"
                                    }
                                }
                            ],
                            "condition": "true"
                        },
                        {
                            "eventName": "State Save",
                            "actions": [
                                {
                                    "setVariable": {
                                        "variableName": "state",
                                        "value": "\"ack\""
                                    }
                                }
                            ],
                            "condition": "true"
                        }
                    ]
                },
                "onExit": {
                    "events": []
                }
            }
        ],
        "initialStateName": "Normal"
    },
    "detectorModelDescription": "This detector model is used to detect if a monitored
 device is in an Alarming State in accordance to the ISA 18.2.",
    "roleArn": "arn:aws:iam::123456789012:role/IoTEventsRole",
    "key": "alarmId"
}
```

例: を使用してシンプルなアラームを構築する AWS IoT Events

このディテクターモデルは、 AWS IoT Events コンソールから利用可能なテンプレートの 1 つです。便宜上、ここに再掲します。

```
{
    "detectorModelDefinition": {
        "states": [
            {
                "onInput": {
                    "transitionEvents": [
                        {
                            "eventName": "not_fixed",
                            "actions": [],
                            "condition": "timeout(\"snoozeTime\")",
                            "nextState": "Alarming"
                        },
                        {
                            "eventName": "reset",
                            "actions": [],
                            "condition":
 "$input.AWS_IoTEvents_Blueprints_Simple_Alarm_Input.command == \"reset\"",
                            "nextState": "Normal"
                        }
                    ],
                    "events": [
                        {
                            "eventName": "DND",
                            "actions": [
                                {
                                    "setVariable": {
                                        "variableName": "dnd_active",
                                        "value": "1"
                                    }
                                }
                            ],
                            "condition":
 "$input.AWS_IoTEvents_Blueprints_Simple_Alarm_Input.command == \"dnd\""
                        }
                    ]
                },
                "stateName": "Snooze",
                "onEnter": {
```

```
                    "events": [
                        {
                            "eventName": "Create Timer",
                            "actions": [
                                {
                                    "setTimer": {
                                        "seconds": 120,
                                        "timerName": "snoozeTime"
                                    }
                                }
                            ],
                            "condition": "true"
                        }
                    ]
                },
                "onExit": {
                    "events": []
                }
            },
            {
                "onInput": {
                    "transitionEvents": [
                        {
                            "eventName": "out_of_range",
                            "actions": [],
                            "condition":
"$input.AWS_IoTEvents_Blueprints_Simple_Alarm_Input.value > $variable.threshold",
                            "nextState": "Alarming"
                        }
                    ],
                    "events": [
                        {
                            "eventName": "Create Config variables",
                            "actions": [
                                {
                                    "setVariable": {
                                        "variableName": "threshold",
                                        "value":
"$input.AWS_IoTEvents_Blueprints_Simple_Alarm_Input.threshold"
                                    }
                                }
                            ],
                            "condition": "$variable.threshold != $variable.threshold"
                        }
```

```
                    ]
                },
                "stateName": "Normal",
                "onEnter": {
                    "events": [
                        {
                            "eventName": "Init",
                            "actions": [
                                {
                                    "setVariable": {
                                        "variableName": "dnd_active",
                                        "value": "0"
                                    }
                                }
                            ],
                            "condition": "true"
                        }
                    ]
                },
                "onExit": {
                    "events": []
                }
            },
            {
                "onInput": {
                    "transitionEvents": [
                        {
                            "eventName": "reset",
                            "actions": [],
                            "condition":
"$input.AWS_IoTEvents_Blueprints_Simple_Alarm_Input.command == \"reset\"",
                            "nextState": "Normal"
                        },
                        {
                            "eventName": "acknowledge",
                            "actions": [],
                            "condition":
"$input.AWS_IoTEvents_Blueprints_Simple_Alarm_Input.command == \"acknowledge\"",
                            "nextState": "Snooze"
                        }
                    ],
                    "events": [
                        {
                            "eventName": "Escalated Alarm Notification",
```

```
                            "actions": [
                                {
                                    "sns": {
                                        "targetArn": "arn:aws:sns:us-
west-2:123456789012:escalatedAlarmNotification"
                                    }
                                }
                            ],
                            "condition": "timeout(\"unacknowledgeTIme\")"
                        }
                    ]
                },
                "stateName": "Alarming",
                "onEnter": {
                    "events": [
                        {
                            "eventName": "Alarm Notification",
                            "actions": [
                                {
                                    "sns": {
                                        "targetArn": "arn:aws:sns:us-
west-2:123456789012:alarmNotification"
                                    }
                                },
                                {
                                    "setTimer": {
                                        "seconds": 300,
                                        "timerName": "unacknowledgeTIme"
                                    }
                                }
                            ],
                            "condition": "$variable.dnd_active != 1"
                        }
                    ]
                },
                "onExit": {
                    "events": []
                }
            }
        ],
        "initialStateName": "Normal"
    },
    "detectorModelDescription": "This detector model is used to detect if a monitored
 device is in an Alarming State.",
```

```
    "roleArn": "arn:aws:iam::123456789012:role/IoTEventsRole",
    "key": "alarmId"
}
```

でのアラームによるモニタリング AWS IoT Events

AWS IoT Events アラームは、データの変更をモニタリングするのに役立ちます。データは、機器とプロセスについて測定するメトリクスにすることができます。しきい値に違反したときに通知を送信するアラームを作成できます。アラームは、問題の検出、メンテナンスの合理化、および機器とプロセスのパフォーマンスの最適化に役立ちます。

アラームは、アラームモデルのインスタンスです。アラームモデルは、何を検出するか、いつ通知を送信するか、誰に通知を受け取るかなどを指定します。アラーム状態が変更されたときに発生する 1 つ以上のサポートされているアクションを指定することもできます。 は、データから派生した入力属性を適切なアラームに AWS IoT Events ルーティングします。モニタリングしているデータが指定された範囲外の場合、アラームが呼び出されます。アラームを確認したり、スヌーズモードに設定したりすることもできます。

の使用 AWS IoT SiteWise

AWS IoT Events アラームを使用して、 のアセットプロパティをモニタリングできます AWS IoT SiteWise。 はアセットプロパティ値を AWS IoT Events アラーム AWS IoT SiteWise に送信します。 はアラーム状態を AWS IoT Events に送信します AWS IoT SiteWise。

AWS IoT SiteWise は外部アラームもサポートしています。の外部でアラームを使用し AWS IoT SiteWise 、アラーム状態データを返すソリューションがある場合は、外部アラームを選択できます。外部アラームには、アラーム状態データを取り込む測定プロパティが含まれています。

AWS IoT SiteWise は外部アラームの状態を評価しません。さらに、アラーム状態が変化したときに外部アラームを確認またはスヌーズすることはできません。

SiteWise モニタリング特徴を使用して、SiteWise モニタリングポータルの外部アラームの状態を表示できます。

詳細については、「AWS IoT SiteWise ユーザーガイド」の「アラームを使用したデータのモニタリング」および「SiteWise モニタリングアプリケーションガイド」の「アラームを使用したモニタリング」を参照してください。

フローの確認

アラームモデルを作成するときに、確認応答フローを有効にするかどうかを選択します。確認応答フローを有効にすると、アラーム状態が変化したときにチームに通知が届きます。チームはアラームを

確認してメモを残すことができます。例えば、アラームの情報と、問題に対処するために実行するアクションを含めることができます。モニタリングしているデータが指定された範囲外の場合、アラームが呼び出されます。

アラームには次の状態があります。

DISABLED

アラームが DISABLED 状態の場合、データを評価する準備ができていません。アラームを有効にするには、アラームを NORMAL 状態に変更する必要があります。

NORMAL

アラームが NORMAL 状態になると、データを評価する準備が整います。

ACTIVE

アラームが ACTIVE 状態の場合、アラームが呼び出されます。モニタリングしているデータが指定範囲外です。

ACKNOWLEDGED

アラームが ACKNOWLEDGED 状態の場合、アラームが呼び出され、アラームを確認しました。

LATCHED

アラームが呼び出されましたが、しばらくしてからアラームを確認しませんでした。アラームは自動的に NORMAL 状態に変わります。

SNOOZE_DISABLED

アラームが SNOOZE_DISABLED 状態の場合、アラームは指定された期間無効になります。スヌーズ時間の後、アラームは自動的に NORMAL 状態に変わります。

でのアラームモデルの作成 AWS IoT Events

AWS IoT Events アラームを使用してデータをモニタリングし、しきい値を超えたときに通知を受け取ることができます。アラームは、アラームモデルを作成または設定するために使用するパラメータを提供します。 AWS IoT Events コンソールまたは AWS IoT Events API を使用して、アラームモデルを作成または設定できます。アラームモデルを設定すると、新しいデータが到着すると変更が有効になります。

要件

アラームモデルを作成する場合は、次の要件が適用されます。

- アラームモデルを作成して、 の入力属性 AWS IoT Events または のアセットプロパティをモニタリングできます AWS IoT SiteWise。
  - アラームモデルを作成するでモデルの入力を作成する AWS IoT Events前に AWS IoT Events、で入力属性をモニタリングすることを選択した場合。
  - アセットプロパティをモニタリングする場合は、アラームモデルを作成する前に でアセットモデルを作成する必要があります。 AWS IoT SiteWise
- アラームがアクションを実行し、 AWS リソースにアクセスできるようにする IAM ロールが必要です。詳細については、「AWS IoT Eventsの許可の設定」を参照してください。
- このチュートリアルで使用するすべての AWS リソースは、同じ AWS リージョンに存在する必要があります。

アラームモデルの作成 (コンソール)

AWS IoT Events コンソールで AWS IoT Events 属性をモニタリングするアラームモデルを作成する方法を以下に示します。

1. AWS IoT Events コンソールにサインインします。
2. ナビゲーションペインで、アラームモデルを選択します。
3. アラームモデルページで、アラームモデルの作成を選択します。
4. アラームモデルの詳細セクションで、次の手順を実行します。
   a. 一意の名前を入力してください。
   b. (オプション) 説明を入力します。
5. アラームターゲットセクションで、次の手順を実行します。

> **Important**
> AWS IoT SiteWise アセットプロパティを選択する場合は、 AWS IoT SiteWiseでアセットモデルを作成しておく必要があります。

   a. AWS IoT Events 入力属性を選択します。

b. 入力を選択します。

c. 入力属性キーを選択します。この入力属性は、アラームを作成するためのキーとして使用されます。このキーに関連付けられた入力をアラームに AWS IoT Events ルーティングします。

> **Important**
>
> 入力メッセージのペイロードにこの入力属性キーが含まれていない場合、またはキーに指定されているの同じ JSON パスにキーが存在しない場合、メッセージは AWS IoT Eventsでの取り込みに失敗します。

6. しきい値の定義セクションでは、 AWS IoT Events がアラームの状態を変更するために使用する入力属性、しきい値、比較演算子を定義します。

   a. 入力属性で、モニタリングする属性を選択します。

      この入力属性が新しいデータを受信するたびに、アラームの状態を判別するために評価されます。

   b. 演算子で、比較演算子を選択します。オペレーターは、入力属性を属性のしきい値と比較します。

      これらのオプションから選択できます。

      - [> greater than] (> より大きい)
      - [>= greater than or equal to] (>= 以上)
      - [< less than)] (< 未満)
      - [<= less than or equal to] (<= 以下)
      - [= equal to] (= 等しい)
      - [!= not equal to] (!= 等しくない)

   c. しきい値には、 AWS IoT Events 入力に数値を入力するか、属性を選択します。 AWS IoT Events はこの値を選択した入力属性の値と比較します。

   d. (オプション) 重要度の場合、このアラームの重要度を反映するためにチームが理解している数値を使用します。

7. (オプション) 通知設定セクションで、アラームの通知設定を設定します。

   最大 10 の通知を追加できます。通知 1 の場合、次のようにします。

a. プロトコルの場合、次のオプションから選択します。

- 電子メールとテキスト - アラームは SMS 通知と電子メール通知を送信します。
- 電子メール - アラームは電子メール通知を送信します。
- テキスト - アラームは SMS 通知を送信します。

b. 送信者には、このアラームに関する通知を送信できる電子メールアドレスを指定します。

送信者リストにさらに電子メールアドレスを追加するには、送信者の追加を選択します。

c. (オプション) 受信者で、受信者を選択します。

受信者リストにユーザーを追加するには、新しいユーザーの追加を選択します。アラームモデルに新しいユーザーを追加する前に、IAM Identity Center ストアに新しいユーザーを追加する必要があります。詳細については、「でアラーム受信者の IAM Identity Center アクセスを管理する AWS IoT Events」を参照してください。

d. (オプション) 追加のカスタムメッセージに、アラームが検出する内容と受信者が実行する必要のあるアクションを説明するメッセージを入力します。

8. インスタンスセクションでは、このアラームモデルに基づいて作成されたすべてのアラームインスタンスを有効または無効にできます。

9. 詳細設定セクションで、次の手順を実行します。

a. 確認フローの場合、通知を有効または無効にできます。

- 有効を選択すると、アラーム状態が変化したときに通知を受け取ります。アラーム状態が正常に戻る前に、通知を確認する必要があります。
- 無効を選択した場合、アクションは不要です。測定値が指定範囲に戻ると、アラームは自動的に通常の状態に変わります。

詳細については、「フローの確認」を参照してください。

b. 許可で、次のいずれかのオプションを選択します。

- AWS ポリシーテンプレートから新しいロールを作成すると、 AWS IoT Events 自動的に IAM ロールが作成されます。
- このアラームモデルがアクションを実行し、他の AWS リソースにアクセスできるようにする既存の IAM ロールを使用できます。

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

c. 追加通知設定では、 AWS Lambda 関数を編集してアラーム通知を管理できます。 AWS Lambda 関数に次のいずれかのオプションを選択します。

- 新しい AWS Lambda 関数を作成する - AWS IoT Events は新しい AWS Lambda 関数を作成します。
- 既存の AWS Lambda 関数を使用する - AWS Lambda 関数名を選択して既存の関数を使用します AWS Lambda 。

可能なアクションの詳細については、「AWS IoT Events 他の AWS サービスの使用」を参照してください。

d. (オプション) 状態の設定アクションでは、アラーム状態が変更されたときに実行する AWS IoT Events アクションを 1 つ以上追加できます。

10. (オプション) タグを追加して、アラームを管理できます。詳細については、「AWS IoT Events リソースのタグ付け」を参照してください。
11. [Create] (作成) を選択します。

でのアラームへの対応 AWS IoT Events

アラームに効果的に応答することは、 で IoT システムを管理する上で重要な側面です AWS IoT Events。通知チャネルの設定、エスカレーション手順の定義、自動応答アクションの実装など、アラームを設定および処理するさまざまな方法について説明します。微妙なアラーム条件を作成し、アラートに優先順位を付け、他の AWS サービスと統合して、IoT アプリケーションの応答性の高いアラーム管理システムを構築する方法について説明します。

確認応答フローを有効にした場合、アラーム状態が変化したときに通知を受け取ります。アラームに応答するには、アラームを確認、無効化、有効化、リセット、またはスヌーズします。

Console

以下に、 AWS IoT Events コンソールでアラームに応答する方法を示します。

1. AWS IoT Events コンソールにサインインします。
2. ナビゲーションペインで、アラームモデルを選択します。

3. ターゲットのアラームモデルを選択します。
4. アラームのリストセクションで、ターゲットアラームを選択します。
5. アクションから次のオプションのいずれかを選択できます。

   - [Acknowledge] (確認) - アラームは、`ACKNOWLEDGED` 状態に変わります。
   - [Disable] (無効) - アラームは、`DISABLED` 状態に変わります。
   - [Enable] (有効) - アラームは、`NORMAL` 状態に変わります。
   - [Reset] (リセット) - アラームは、`NORMAL` 状態に変わります。
   - [Snooze] (スヌーズ) を選択して、以下を行う。

     1. スヌーズの長さを選択するか、カスタムスヌーズの長さを入力します。
     2. [保存] を選択します。

     アラームは、`SNOOZE_DISABLED` 状態になります。

   アラーム状態の詳細については、「フローの確認」を参照してください。

API

1 つ以上のアラームに応答するには、次の AWS IoT Events API オペレーションを使用できます。

- BatchAcknowledgeAlarm
- BatchDisableAlarm
- BatchEnableAlarm
- BatchResetAlarm
- BatchSnoozeAlarm

でのアラーム通知の管理 AWS IoT Events

AWS IoT Events は Lambda と統合され、カスタムイベント処理機能を提供します。このセクションでは、ディ AWS IoT Events テクターモデル内で Lambda 関数を使用する方法について説明します。これにより、複雑なロジックの実行、外部サービスとの対話、高度なイベント処理の実装が可能になります。

AWS IoT Events は Lambda 関数を使用してアラーム通知を管理します。が提供する Lambda 関数を使用する AWS IoT Events か、新しい関数を作成できます。

トピック

- での Lambda 関数の作成 AWS IoT Events
- AWS IoT Eventsが提供する Lambda 関数を使用する
- でアラーム受信者の IAM Identity Center アクセスを管理する AWS IoT Events

での Lambda 関数の作成 AWS IoT Events

AWS IoT Events は、アラームが E メールおよび SMS 通知を送受信できるようにする Lambda 関数を提供します。

要件

アラームの Lambda 関数を作成する場合は、次の要件が適用されます。

- アラームが SMS 通知を送信する場合は、Amazon SNS が SMS メッセージを配信するように設定されていることを確認します。
  - 詳細については、次のドキュメントを参照してください。
    - Amazon SNS Simple Notification Service デベロッパーガイド」の「Amazon SNS および Amazon SNS SMS メッセージの送信元 ID を使用したモバイルテキストメッセージ」。Amazon SNS
    - AWS SMS 「 ユーザーガイド」の「AWS エンドユーザーメッセージング SMS とは」。
- アラームが E メールまたは SMS 通知を送信する場合は、 が Amazon SES および Amazon SNS と連携 AWS Lambda できるようにする IAM ロールが必要です。

  ポリシーの例:

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

```
                "ses:SendEmail",
                "ses:VerifyEmailIdentity"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "sns:Publish",
                "sns:OptInPhoneNumber",
                "sns:CheckIfPhoneNumberIsOptedOut",
                "sms-voice:DescribeOptedOutNumbers"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Deny",
            "Action": "sns:Publish",
            "Resource": "arn:aws:sns:*:*:*"
        },
        {
          "Effect" : "Allow",
          "Action" : [
            "logs:CreateLogGroup",
            "logs:CreateLogStream",
            "logs:PutLogEvents"
          ],
          "Resource" : "*"
        }
    ]
}
```

- AWS IoT Events と の両方に同じ AWS リージョンを選択する必要があります AWS Lambda。サポートされているリージョンのリストについては、「Amazon Web Services 全般のリファレンス」の「AWS IoT Events のエンドポイントとクォータ」および[AWS Lambda のエンドポイントとクォータ」を参照してください。

を使用するための AWS IoT Events Lambda 関数をデプロイする AWS CloudFormation

このチュートリアルでは、 AWS CloudFormation テンプレートを使用して Lambda 関数をデプロイします。このテンプレートは、Lambda 関数が Amazon SES および Amazon SNS と連携できるようにする IAM ロールを自動的に作成します。

以下は、 AWS Command Line Interface （ AWS CLI) を使用して CloudFormation スタックを作成する方法を示しています。

1. デバイスのターミナルで、 `aws --version`を実行して、 をインストールしたかどうかを確認します AWS CLI。詳細については、「AWS Command Line Interface ユーザーガイド」の「AWS CLI の最新バージョンをインストールまたは更新」を参照してください。
2. を実行して`aws configure list`、このチュートリアルのすべての AWS リソースがある AWS リージョン AWS CLI で を設定しているかどうかを確認します。詳細については、「 AWS Command Line Interface ユーザーガイド」の「 コマンドを使用して構成設定を設定および表示する」を参照してください。
3. CloudFormation テンプレート notificationLambda.template.yaml.zip をダウンロードします。

   > **Note**
   > ファイルのダウンロードに問題がある場合は、CloudFormation テンプレート でもテンプレートを利用できます。

4. コンテンツを解凍し、`notificationLambda.template.yaml` としてローカルに保存します。
5. デバイスでターミナルを開き、`notificationLambda.template.yaml` ファイルをダウンロードしたディレクトリに移動します。
6. CloudFormation スタックを作成するには、次のコマンドを実行します。

```
aws cloudformation create-stack --stack-name notificationLambda-stack --template-
body file://notificationLambda.template.yaml --capabilities CAPABILITY_IAM
```

この CloudFormation テンプレートを変更して、Lambda 関数とその動作をカスタマイズできます。

> **Note**
>
> AWS Lambda は関数エラーを 2 回再試行します。関数にすべての着信要求を処理するのに十分な容量がない場合、イベントはキュー内で数時間または数日待って関数に送信される可能性があります。正常に処理されなかったイベントをキャプチャするように、関数に未配信メッセージキュー (DLQ) を設定できます。詳細については、「AWS Lambda デベロッパーガイド」の「非同期呼び出し」を参照してください。

CloudFormation コンソールでスタックを作成または設定することもできます。詳細については、「AWS CloudFormation ユーザーガイド」の「スタックの操作」を参照してください。

のカスタム Lambda 関数の作成 AWS IoT Events

Lambda 関数を作成するか、 AWS IoT Eventsが提供する関数を変更できます。

カスタム Lambda 関数を作成する場合は、次の要件が適用されます。

- Lambda 関数が指定されたアクションを実行し、 AWS リソースにアクセスできるようにするアクセス許可を追加します。
- が提供する Lambda 関数を使用する場合は AWS IoT Events、必ず Python 3.7 ランタイムを選択してください。

Lambda 関数の例:

```
import boto3
import json
import logging
import datetime
logger = logging.getLogger()
logger.setLevel(logging.INFO)
ses = boto3.client('ses')
sns = boto3.client('sns')
def check_value(target):
  if target:
    return True
  return False

# Check whether email is verified. Only verified emails are allowed to send emails to
 or from.
```

```
def check_email(email):
  if not check_value(email):
    return False
  result = ses.get_identity_verification_attributes(Identities=[email])
  attr = result['VerificationAttributes']
  if (email not in attr or attr[email]['VerificationStatus'] != 'Success'):
      logging.info('Verification email for {} sent. You must have all the emails
 verified before sending email.'.format(email))
      ses.verify_email_identity(EmailAddress=email)
      return False
  return True

# Check whether the phone holder has opted out of receiving SMS messages from your
 account
def check_phone_number(phone_number):
  try:
    result = sns.check_if_phone_number_is_opted_out(phoneNumber=phone_number)
    if (result['isOptedOut']):
        logger.info('phoneNumber {} is not opt in of receiving SMS messages. Phone
 number must be opt in first.'.format(phone_number))
        return False
    return True
  except Exception as e:
    logging.error('Your phone number {} must be in E.164 format in SSO. Exception
 thrown: {}'.format(phone_number, e))
    return False

def check_emails(emails):
  result = True
  for email in emails:
      if not check_email(email):
          result = False
  return result

def lambda_handler(event, context):
  logging.info('Received event: ' + json.dumps(event))
  nep = json.loads(event.get('notificationEventPayload'))
  alarm_state = nep['alarmState']
  default_msg = 'Alarm ' + alarm_state['stateName'] + '\n'
  timestamp =
 datetime.datetime.utcfromtimestamp(float(nep['stateUpdateTime'])/1000).strftime('%Y-
%m-%d %H:%M:%S')
  alarm_msg = "{} {} {} at {} UTC ".format(nep['alarmModelName'], nep.get('keyValue',
 'Singleton'), alarm_state['stateName'], timestamp)
```

```
  default_msg += 'Sev: ' + str(nep['severity']) + '\n'
  if (alarm_state['ruleEvaluation']):
    property = alarm_state['ruleEvaluation']['simpleRule']['inputProperty']
    default_msg += 'Current Value: ' + str(property) + '\n'
    operator = alarm_state['ruleEvaluation']['simpleRule']['operator']
    threshold = alarm_state['ruleEvaluation']['simpleRule']['threshold']
    alarm_msg += '({} {} {})'.format(str(property), operator, str(threshold))
  default_msg += alarm_msg + '\n'

  emails = event.get('emailConfigurations', [])
  logger.info('Start Sending Emails')
  for email in emails:
    from_adr = email.get('from')
    to_adrs = email.get('to', [])
    cc_adrs = email.get('cc', [])
    bcc_adrs = email.get('bcc', [])
    msg = default_msg + '\n' + email.get('additionalMessage', '')
    subject = email.get('subject', alarm_msg)
    fa_ver = check_email(from_adr)
    tas_ver = check_emails(to_adrs)
    ccas_ver = check_emails(cc_adrs)
    bccas_ver = check_emails(bcc_adrs)
    if (fa_ver and tas_ver and ccas_ver and bccas_ver):
      ses.send_email(Source=from_adr,
                     Destination={'ToAddresses': to_adrs, 'CcAddresses': cc_adrs,
 'BccAddresses': bcc_adrs},
                     Message={'Subject': {'Data': subject}, 'Body': {'Text': {'Data':
 msg}}})
      logger.info('Emails have been sent')

  logger.info('Start Sending SNS message to SMS')
  sns_configs = event.get('smsConfigurations', [])
  for sns_config in sns_configs:
    sns_msg = default_msg + '\n' + sns_config.get('additionalMessage', '')
    phone_numbers = sns_config.get('phoneNumbers', [])
    sender_id = sns_config.get('senderId')
    for phone_number in phone_numbers:
        if check_phone_number(phone_number):
          if check_value(sender_id):
            sns.publish(PhoneNumber=phone_number, Message=sns_msg,
 MessageAttributes={'AWS.SNS.SMS.SenderID':{'DataType': 'String','StringValue':
 sender_id}})
          else:
            sns.publish(PhoneNumber=phone_number, Message=sns_msg)
```

```
        logger.info('SNS messages have been sent')
```

詳細については、「AWS Lambda デベロッパーガイド」の「AWS Lambdaとは」を参照してください。

CloudFormation テンプレート

次の CloudFormation テンプレートを使用して、Lambda 関数を作成します。

```
AWSTemplateFormatVersion: '2010-09-09'
Description: 'Notification Lambda for Alarm Model'
Resources:
  NotificationLambdaRole:
    Type: AWS::IAM::Role
    Properties:
      AssumeRolePolicyDocument:
        Statement:
          - Effect: Allow
            Principal:
              Service: lambda.amazonaws.com
            Action: sts:AssumeRole
      Path: "/"
      ManagedPolicyArns:
        - 'arn:aws:iam::aws:policy/AWSLambdaExecute'
      Policies:
        - PolicyName: "NotificationLambda"
          PolicyDocument:
            Version: "2012-10-17"
            Statement:
              - Effect: "Allow"
                Action:
                  - "ses:GetIdentityVerificationAttributes"
                  - "ses:SendEmail"
                  - "ses:VerifyEmailIdentity"
                Resource: "*"
              - Effect: "Allow"
                Action:
                  - "sns:Publish"
                  - "sns:OptInPhoneNumber"
                  - "sns:CheckIfPhoneNumberIsOptedOut"
                  - "sms-voice:DescribeOptedOutNumbers"
                Resource: "*"
              - Effect: "Deny"
```

```
                Action:
                  - "sns:Publish"
                Resource: "arn:aws:sns:*:*:*"
 NotificationLambdaFunction:
   Type: AWS::Lambda::Function
   Properties:
     Role: !GetAtt NotificationLambdaRole.Arn
     Runtime: python3.7
     Handler: index.lambda_handler
     Timeout: 300
     MemorySize: 3008
     Code:
       ZipFile: |
         import boto3
         import json
         import logging
         import datetime
         logger = logging.getLogger()
         logger.setLevel(logging.INFO)
         ses = boto3.client('ses')
         sns = boto3.client('sns')
         def check_value(target):
           if target:
             return True
           return False

         # Check whether email is verified. Only verified emails are allowed to send
 emails to or from.
         def check_email(email):
           if not check_value(email):
             return False
           result = ses.get_identity_verification_attributes(Identities=[email])
           attr = result['VerificationAttributes']
           if (email not in attr or attr[email]['VerificationStatus'] != 'Success'):
               logging.info('Verification email for {} sent. You must have all the
 emails verified before sending email.'.format(email))
               ses.verify_email_identity(EmailAddress=email)
               return False
           return True

         # Check whether the phone holder has opted out of receiving SMS messages from
 your account
         def check_phone_number(phone_number):
           try:
```

```
              result = sns.check_if_phone_number_is_opted_out(phoneNumber=phone_number)
              if (result['isOptedOut']):
                  logger.info('phoneNumber {} is not opt in of receiving SMS messages.
 Phone number must be opt in first.'.format(phone_number))
                  return False
              return True
            except Exception as e:
              logging.error('Your phone number {} must be in E.164 format in SSO.
 Exception thrown: {}'.format(phone_number, e))
              return False

          def check_emails(emails):
            result = True
            for email in emails:
                if not check_email(email):
                    result = False
            return result

          def lambda_handler(event, context):
            logging.info('Received event: ' + json.dumps(event))
            nep = json.loads(event.get('notificationEventPayload'))
            alarm_state = nep['alarmState']
            default_msg = 'Alarm ' + alarm_state['stateName'] + '\n'
            timestamp =
 datetime.datetime.utcfromtimestamp(float(nep['stateUpdateTime'])/1000).strftime('%Y-
%m-%d %H:%M:%S')
            alarm_msg = "{} {} {} at {} UTC ".format(nep['alarmModelName'],
 nep.get('keyValue', 'Singleton'), alarm_state['stateName'], timestamp)
            default_msg += 'Sev: ' + str(nep['severity']) + '\n'
            if (alarm_state['ruleEvaluation']):
              property = alarm_state['ruleEvaluation']['simpleRule']['inputProperty']
              default_msg += 'Current Value: ' + str(property) + '\n'
              operator = alarm_state['ruleEvaluation']['simpleRule']['operator']
              threshold = alarm_state['ruleEvaluation']['simpleRule']['threshold']
              alarm_msg += '({} {} {})'.format(str(property), operator, str(threshold))
            default_msg += alarm_msg + '\n'

            emails = event.get('emailConfigurations', [])
            logger.info('Start Sending Emails')
            for email in emails:
              from_adr = email.get('from')
              to_adrs = email.get('to', [])
              cc_adrs = email.get('cc', [])
              bcc_adrs = email.get('bcc', [])
```

```
              msg = default_msg + '\n' + email.get('additionalMessage', '')
              subject = email.get('subject', alarm_msg)
              fa_ver = check_email(from_adr)
              tas_ver = check_emails(to_adrs)
              ccas_ver = check_emails(cc_adrs)
              bccas_ver = check_emails(bcc_adrs)
              if (fa_ver and tas_ver and ccas_ver and bccas_ver):
                ses.send_email(Source=from_adr,
                               Destination={'ToAddresses': to_adrs, 'CcAddresses':
cc_adrs, 'BccAddresses': bcc_adrs},
                               Message={'Subject': {'Data': subject}, 'Body': {'Text':
{'Data': msg}}})
                logger.info('Emails have been sent')

            logger.info('Start Sending SNS message to SMS')
            sns_configs = event.get('smsConfigurations', [])
            for sns_config in sns_configs:
              sns_msg = default_msg + '\n' + sns_config.get('additionalMessage', '')
              phone_numbers = sns_config.get('phoneNumbers', [])
              sender_id = sns_config.get('senderId')
              for phone_number in phone_numbers:
                  if check_phone_number(phone_number):
                    if check_value(sender_id):
                      sns.publish(PhoneNumber=phone_number, Message=sns_msg,
MessageAttributes={'AWS.SNS.SMS.SenderID':{'DataType': 'String','StringValue':
sender_id}})
                    else:
                      sns.publish(PhoneNumber=phone_number, Message=sns_msg)
                    logger.info('SNS messages have been sent')
```

AWS IoT Eventsが提供する Lambda 関数を使用する

アラーム通知では、アラーム通知を管理する AWS IoT Events ために が提供する Lambda 関数を使用できます。

AWS IoT Events が提供する Lambda 関数を使用してアラーム通知を管理する場合は、次の要件が適用されます。

- Amazon Simple Email Service (Amazon SES) で E メール通知を送信する E メールアドレスを確認する必要があります。詳細については、「Amazon Simple Email Service デベロッパーガイド」の「E メールアドレスのアイデンティティの確認」を参照してください。

検証リンクを受け取ったら、リンクをクリックしてメールアドレスを確認します。スパムフォルダで検証メールをチェックすることもできます。

- アラームが SMS 通知を送信する場合は、電話番号に E.164 国際電話番号のフォーマットを使用する必要があります。このフォーマットには `+<country-calling-code><area-code><phone-number>` が含まれています。

  電話番号の例:

国	市内電話番号	E.164 形式の番号
アメリカ	206-555-0100	+12065550100
英国	020-1234-1234	+442012341234
リトアニア	8+601+12345	+37060112345

  国番号を見つけるには、countrycode.org にアクセスしてください。

  が提供する Lambda 関数 AWS IoT Events は、E.164 形式の電話番号を使用しているかどうかを確認します。ただし、電話番号は確認されません。正確な電話番号を入力したが SMS 通知を受信しなかったことを確認した場合は、電話会社に連絡することができます。キャリアはメッセージをブロックする可能性があります。

でアラーム受信者の IAM Identity Center アクセスを管理する AWS IoT Events

AWS IoT Events は AWS IAM Identity Center を使用して、アラーム受信者の SSO アクセスを管理します。 AWS IoT Events 通知受信者に IAM アイデンティティセンターを実装すると、セキュリティとユーザーエクスペリエンスが向上します。アラームが受信者に通知を送信できるようにするには、IAM Identity Center を有効にして、受信者を IAM Identity Center ストアに追加する必要があります。詳細については、「AWS IAM Identity Center ユーザーガイド」の「ユーザーの追加」を参照してください。

Important

- AWS IoT Events AWS Lambda、、および IAM Identity Center には同じ AWS リージョンを選択する必要があります。
- AWS Organizations は、一度に 1 つの IAM アイデンティティセンターリージョンのみをサポートします。別のリージョンで IAM Identity Center を利用できるようにする場合は、最初に現在の IAM Identity Center 設定を削除する必要があります。詳細については、「AWS IAM Identity Center ユーザーガイド」の「IAM Identity Center のリージョンデータ」を参照してください。

のセキュリティ AWS IoT Events

のクラウドセキュリティが最優先事項 AWS です。 AWS のお客様は、セキュリティを最も重視する組織の要件を満たすように構築されたデータセンターとネットワークアーキテクチャからメリットを得られます。

セキュリティは、 AWS とお客様の間で共有される責任です。責任共有モデルでは、この責任がクラウドのセキュリティおよびクラウド内のセキュリティとして説明されています。

- クラウドのセキュリティ – AWS クラウドで AWS サービスを実行するインフラストラクチャを保護する AWS 責任があります。 AWS また、 は、お客様が安全に使用できるサービスも提供します。セキュリティの有効性は、AWS コンプライアンスプログラムの一環として、サードパーティーの審査機関によって定期的にテストおよび検証されています。が適用されるコンプライアンスプログラムの詳細については AWS IoT Events、AWS 「コンプライアンスプログラムによる対象範囲内のサービス」を参照してください。
- クラウド内のセキュリティ – お客様の責任は、使用する AWS サービスによって決まります。また、お客様は、お客様のデータの機密性、組織の要件、および適用可能な法律および規制などの他の要因についても責任を担います。

このドキュメントは、 を使用する際の責任共有モデルの適用方法を理解するのに役立ちます AWS IoT Events。以下のトピックでは、セキュリティおよびコンプライアンスの目的を達成する AWS IoT Events ように を設定する方法について説明します。また、 AWS IoT Events リソースのモニタリングや保護に役立つ他の AWS サービスの使用方法についても説明します。

トピック

- の Identity and Access Management AWS IoT Events
- 信頼性、可用性、パフォーマンス AWS IoT Events を維持するためのモニタリング
- のコンプライアンス検証 AWS IoT Events
- の耐障害性 AWS IoT Events
- のインフラストラクチャセキュリティ AWS IoT Events

の Identity and Access Management AWS IoT Events

AWS Identity and Access Management (IAM) は、管理者が AWS リソースへのアクセスを安全に制御するのに役立つ AWS サービスです。IAM 管理者は、誰を認証 (サインイン) し、誰に AWS IoT

Events リソースの使用を承認する (アクセス許可を付与する) かを制御します。IAM は、追加料金なしで使用できる AWS サービスです。

トピック

- 対象者
- アイデンティティを使用した認証
- ポリシーを使用したアクセスの管理
- ID とアクセス管理の詳細
- と IAM の AWS IoT Events 連携方法
- AWS IoT Events アイデンティティベースのポリシーの例
- のサービス間の混乱した代理の防止 AWS IoT Events
- AWS IoT Events ID とアクセスのトラブルシューティング

対象者

AWS Identity and Access Management (IAM) の使用方法は、作業内容によって異なります AWS IoT Events。

サービスユーザー – AWS IoT Events サービスを使用してジョブを実行する場合、管理者から必要な認証情報とアクセス許可が与えられます。さらに多くの AWS IoT Events 機能を使用して作業を行う場合は、追加のアクセス許可が必要になることがあります。アクセスの管理方法を理解すると、管理者から適切な権限をリクエストするのに役に立ちます。 AWS IoT Events機能にアクセスできない場合は、「AWS IoT Events ID とアクセスのトラブルシューティング」を参照してください。

サービス管理者 – 社内の AWS IoT Events リソースを担当している場合は、通常、 へのフルアクセスがあります AWS IoT Events。サービスユーザーがどの AWS IoT Events 機能やリソースにアクセスするかを決めるのは管理者の仕事です。その後、IAM 管理者にリクエストを送信して、サービスユーザーの権限を変更する必要があります。このページの情報を点検して、IAM の基本概念を理解してください。会社で IAM を で使用する方法の詳細については AWS IoT Events、「」を参照してくださいと IAM の AWS IoT Events 連携方法。

IAM 管理者 - 管理者は、 AWS IoT Eventsへのアクセスを管理するポリシーの書き込み方法の詳細について確認する場合があります。IAM で使用できる AWS IoT Events アイデンティティベースのポリシーの例を表示するには、「」を参照してくださいAWS IoT Events アイデンティティベースのポリシーの例。

アイデンティティを使用した認証

認証とは、ID 認証情報 AWS を使用して にサインインする方法です。として、IAM ユーザーとして AWS アカウントのルートユーザー、または IAM ロールを引き受けて認証 (サインイン AWS) される必要があります。

ID ソースを介して提供された認証情報を使用して、フェデレーティッド ID AWS として にサインインできます。 AWS IAM Identity Center（ IAM Identity Center) ユーザー、会社のシングルサインオン認証、Google または Facebook 認証情報は、フェデレーティッド ID の例です。フェデレーティッド ID としてサインインする場合、IAM ロールを使用して、前もって管理者により ID フェデレーションが設定されています。フェデレーションを使用して にアクセスすると、間接的 AWS にロールを引き受けます。

ユーザーのタイプに応じて、 AWS Management Console または AWS アクセスポータルにサインインできます。へのサインインの詳細については AWS、「 AWS サインイン ユーザーガイド」の「 にサインインする方法 AWS アカウント」を参照してください。

AWS プログラムで にアクセスする場合、 は Software Development Kit (SDK) とコマンドラインインターフェイス (CLI) AWS を提供し、認証情報を使用してリクエストに暗号で署名します。 AWS ツールを使用しない場合は、リクエストに自分で署名する必要があります。リクエストに自分で署名する推奨方法の使用については、「IAM ユーザーガイド」の「API リクエストに対するAWS Signature Version 4」を参照してください。

使用する認証方法を問わず、追加セキュリティ情報の提供をリクエストされる場合もあります。たとえば、 では、アカウントのセキュリティを強化するために多要素認証 (MFA) を使用する AWS ことをお勧めします。詳細については、「AWS IAM Identity Center ユーザーガイド」の「多要素認証」および「IAM ユーザーガイド」の「IAM のAWS 多要素認証」を参照してください。

AWS アカウント ルートユーザー

を作成するときは AWS アカウント、アカウント内のすべての およびリソースへの AWS のサービス完全なアクセス権を持つ 1 つのサインインアイデンティティから始めます。この ID は AWS アカウント ルートユーザーと呼ばれ、アカウントの作成に使用した E メールアドレスとパスワードでサインインすることでアクセスできます。日常的なタスクには、ルートユーザーを使用しないことを強くお勧めします。ルートユーザーの認証情報は保護し、ルートユーザーでしか実行できないタスクを実行するときに使用します。ルートユーザーとしてサインインする必要があるタスクの完全なリストについては、IAM ユーザーガイドの「ルートユーザー認証情報が必要なタスク」を参照してください。

IAM ユーザーとグループ

IAM ユーザーは、1 人のユーザーまたはアプリケーションに対して特定のアクセス許可 AWS アカウント を持つ 内のアイデンティティです。可能であれば、パスワードやアクセスキーなどの長期的な認証情報を保有する IAM ユーザーを作成する代わりに、一時的な認証情報を使用することをお勧めします。ただし、IAM ユーザーでの長期的な認証情報が必要な特定のユースケースがある場合は、アクセスキーをローテーションすることをお勧めします。詳細については、「IAM ユーザーガイド」の「長期的な認証情報を必要とするユースケースのためにアクセスキーを定期的にローテーションする」を参照してください。

IAM グループは、IAM ユーザーの集団を指定するアイデンティティです。グループとしてサインインすることはできません。グループを使用して、複数のユーザーに対して一度に権限を指定できます。多数のユーザーグループがある場合、グループを使用することで権限の管理が容易になります。例えば、IAMAdmins という名前のグループを設定して、そのグループに IAM リソースを管理する許可を与えることができます。

ユーザーは、ロールとは異なります。ユーザーは 1 人の人または 1 つのアプリケーションに一意に関連付けられますが、ロールはそれを必要とする任意の人が引き受けるようになっています。ユーザーには永続的な長期の認証情報がありますが、ロールでは一時認証情報が提供されます。詳細については、「IAM ユーザーガイド」の「IAM ユーザーのユースケース」を参照してください。

IAM ロール

IAM ロールは、特定のアクセス許可 AWS アカウント を持つ 内のアイデンティティです。これは IAM ユーザーに似ていますが、特定のユーザーには関連付けられていません。で IAM ロールを一時的に引き受けるには AWS Management Console、ユーザーから IAM ロール (コンソール) に切り替えることができます。ロールを引き受けるには、 または AWS API オペレーションを AWS CLI 呼び出すか、カスタム URL を使用します。ロールを使用する方法の詳細については、「IAM ユーザーガイド」の「ロールを引き受けるための各種方法」を参照してください。

IAM ロールと一時的な認証情報は、次の状況で役立ちます:

- フェデレーションユーザーアクセス – フェデレーティッド ID に許可を割り当てるには、ロールを作成してそのロールの許可を定義します。フェデレーティッド ID が認証されると、その ID はロールに関連付けられ、ロールで定義されている許可が付与されます。フェデレーションのロールについては、「IAM ユーザーガイド」の「サードパーティー ID プロバイダー (フェデレーション) 用のロールを作成する」を参照してください。IAM Identity Center を使用する場合は、許可セットを設定します。アイデンティティが認証後にアクセスできるものを制御するため、IAM Identity

Center は、権限セットを IAM のロールに関連付けます。アクセス許可セットの詳細については、「AWS IAM Identity Center User Guide」の「Permission sets」を参照してください。

- 一時的な IAM ユーザー権限 - IAM ユーザーまたはロールは、特定のタスクに対して複数の異なる権限を一時的に IAM ロールで引き受けることができます。
- クロスアカウントアクセス - IAM ロールを使用して、自分のアカウントのリソースにアクセスすることを、別のアカウントの人物 (信頼済みプリンシパル) に許可できます。クロスアカウントアクセス権を付与する主な方法は、ロールを使用することです。ただし、一部の では AWS のサービス、 (ロールをプロキシとして使用する代わりに) リソースに直接ポリシーをアタッチできます。クロスアカウントアクセスにおけるロールとリソースベースのポリシーの違いについては、「IAM ユーザーガイド」の「IAM でのクロスアカウントのリソースへのアクセス」を参照してください。
- クロスサービスアクセス — 一部の では、他の の機能 AWS のサービス を使用します AWS のサービス。例えば、あるサービスで呼び出しを行うと、通常そのサービスによって Amazon EC2 でアプリケーションが実行されたり、Amazon S3 にオブジェクトが保存されたりします。サービスでは、呼び出し元プリンシパルの許可、サービスロール、またはサービスリンクロールを使用してこれを行う場合があります。
  - 転送アクセスセッション (FAS) – IAM ユーザーまたはロールを使用してアクションを実行すると AWS、プリンシパルと見なされます。一部のサービスを使用する際に、アクションを実行することで、別のサービスの別のアクションがトリガーされることがあります。FAS は、 を呼び出すプリンシパルのアクセス許可と AWS のサービス、ダウンストリームサービス AWS のサービス へのリクエストのリクエストをリクエストする を組み合わせて使用します。FAS リクエストは、サービスが他の AWS のサービス またはリソースとのやり取りを完了する必要があるリクエストを受け取った場合にのみ行われます。この場合、両方のアクションを実行するためのアクセス許可が必要です。FAS リクエストを行う際のポリシーの詳細については、「転送アクセスセッション」を参照してください。
  - サービスロール - サービスがユーザーに代わってアクションを実行するために引き受ける IAM ロールです。IAM 管理者は、IAM 内からサービスロールを作成、変更、削除することができます。詳細については、「IAM ユーザーガイド」の「AWS のサービスに許可を委任するロールを作成する」を参照してください。
  - サービスにリンクされたロール – サービスにリンクされたロールは、 にリンクされたサービスロールの一種です AWS のサービス。サービスは、ユーザーに代わってアクションを実行するロールを引き受けることができます。サービスにリンクされたロールは に表示され AWS アカウント 、 サービスによって所有されます。IAM 管理者は、サービスリンクロールのアクセス許可を表示できますが、編集することはできません。

- Amazon EC2 で実行されているアプリケーション – IAM ロールを使用して、EC2 インスタンスで実行され、 AWS CLI または AWS API リクエストを実行しているアプリケーションの一時的な認証情報を管理できます。これは、EC2 インスタンス内でのアクセスキーの保存に推奨されます。AWS ロールを EC2 インスタンスに割り当て、そのすべてのアプリケーションで使用できるようにするには、インスタンスにアタッチされたインスタンスプロファイルを作成します。インスタンスプロファイルにはロールが含まれ、EC2 インスタンスで実行されるプログラムは一時的な認証情報を取得できます。詳細については、「IAM ユーザーガイド」の「Amazon EC2 インスタンスで実行されるアプリケーションに IAM ロールを使用して許可を付与する」を参照してください。

ポリシーを使用したアクセスの管理

でアクセスを制御する AWS には、ポリシーを作成し、ID AWS またはリソースにアタッチします。ポリシーは のオブジェクト AWS であり、アイデンティティまたはリソースに関連付けられると、そのアクセス許可を定義します。 は、プリンシパル (ユーザー、ルートユーザー、またはロールセッション) がリクエストを行うときに、これらのポリシー AWS を評価します。ポリシーでの権限により、リクエストが許可されるか拒否されるかが決まります。ほとんどのポリシーは JSON ドキュメント AWS として に保存されます。JSON ポリシードキュメントの構造と内容の詳細については、IAM ユーザーガイドの JSON ポリシー概要を参照してください。

管理者は JSON AWS ポリシーを使用して、誰が何にアクセスできるかを指定できます。つまり、どのプリンシパルがどのリソースに対してどのような条件下でアクションを実行できるかということです。

デフォルトでは、ユーザーやロールに権限はありません。IAM 管理者は、リソースで必要なアクションを実行するための権限をユーザーに付与する IAM ポリシーを作成できます。その後、管理者はロールに IAM ポリシーを追加し、ユーザーはロールを引き受けることができます。

IAM ポリシーは、オペレーションの実行方法を問わず、アクションの許可を定義します。例えば、`iam:GetRole` アクションを許可するポリシーがあるとします。そのポリシーを持つユーザーは、 AWS Management Console、、 AWS CLIまたは AWS API からロール情報を取得できます。

アイデンティティベースのポリシー

アイデンティティベースポリシーは、IAM ユーザーグループ、ユーザーのグループ、ロールなど、アイデンティティにアタッチできる JSON 許可ポリシードキュメントです。これらのポリシーは、ユーザーとロールが実行できるアクション、リソース、および条件をコントロールします。ID ベースのポリシーを作成する方法については、「IAM ユーザーガイド」の「カスタマー管理ポリシーでカスタム IAM アクセス許可を定義する」を参照してください。

アイデンティティベースのポリシーは、さらにインラインポリシーまたはマネージドポリシーに分類できます。インラインポリシーは、単一のユーザー、グループ、またはロールに直接埋め込まれています。管理ポリシーは、 内の複数のユーザー、グループ、ロールにアタッチできるスタンドアロンポリシーです AWS アカウント。管理ポリシーには、 AWS 管理ポリシーとカスタマー管理ポリシーが含まれます。マネージドポリシーまたはインラインポリシーのいずれかを選択する方法については、「IAM ユーザーガイド」の「管理ポリシーとインラインポリシーのいずれかを選択する」を参照してください。

その他のポリシータイプ

AWS は、一般的ではない追加のポリシータイプをサポートしています。これらのポリシータイプでは、より一般的なポリシータイプで付与された最大の権限を設定できます。

- アクセス許可の境界 - アクセス許可の境界は、アイデンティティベースポリシーによって IAM エンティティ (IAM ユーザーまたはロール) に付与できる権限の上限を設定する高度な機能です。エンティティにアクセス許可の境界を設定できます。結果として得られる権限は、エンティティのアイデンティティベースポリシーとそのアクセス許可の境界の共通部分になります。`Principal` フィールドでユーザーまたはロールを指定するリソースベースのポリシーでは、アクセス許可の境界は制限されません。これらのポリシーのいずれかを明示的に拒否した場合、権限は無効になります。アクセス許可の境界の詳細については、「IAM ユーザーガイド」の「IAM エンティティのアクセス許可の境界」を参照してください。
- サービスコントロールポリシー (SCPs) – SCPsは、 の組織または組織単位 (OU) の最大アクセス許可を指定する JSON ポリシーです AWS Organizations。 AWS Organizations は、ビジネスが所有する複数の AWS アカウント をグループ化して一元管理するためのサービスです。組織内のすべての機能を有効にすると、サービスコントロールポリシー (SCP) を一部またはすべてのアカウントに適用できます。SCP は、各 を含むメンバーアカウントのエンティティのアクセス許可を制限します AWS アカウントのルートユーザー。Organizations と SCP の詳細については、「AWS Organizations ユーザーガイド」の「サービスコントロールポリシー (SCP)」を参照してください。
- リソースコントロールポリシー (RCP) – RCP は、所有する各リソースにアタッチされた IAM ポリシーを更新することなく、アカウント内のリソースに利用可能な最大数のアクセス許可を設定するために使用できる JSON ポリシーです。RCP は、メンバーアカウントのリソースのアクセス許可を制限し、組織に属しているかどうかにかかわらず AWS アカウントのルートユーザー、 を含む ID の有効なアクセス許可に影響を与える可能性があります。RCP をサポートする のリストを含む Organizations と RCP の詳細については、AWS Organizations RCPs「リソースコントロールポリシー (RCPs」を参照してください。 AWS のサービス

- セッションポリシー - セッションポリシーは、ロールまたはフェデレーションユーザーの一時的なセッションをプログラムで作成する際にパラメータとして渡す高度なポリシーです。結果としてセッションの権限は、ユーザーまたはロールのアイデンティティベースポリシーとセッションポリシーの共通部分になります。また、リソースベースのポリシーから権限が派生する場合もあります。これらのポリシーのいずれかを明示的に拒否した場合、権限は無効になります。詳細については、「IAM ユーザーガイド」の「セッションポリシー」を参照してください。

複数のポリシータイプ

1 つのリクエストに複数のタイプのポリシーが適用されると、結果として作成される権限を理解するのがさらに難しくなります。複数のポリシータイプが関係する場合に がリクエストを許可するかどうか AWS を決定する方法については、「IAM ユーザーガイド」の「ポリシー評価ロジック」を参照してください。

ID とアクセス管理の詳細

の ID とアクセスの管理の詳細については AWS IoT Events、次のページを参照してください。

- と IAM の AWS IoT Events 連携方法
- AWS IoT Events ID とアクセスのトラブルシューティング

と IAM の AWS IoT Events 連携方法

IAM を使用して へのアクセスを管理する前に AWS IoT Events、 で使用できる IAM 機能を理解しておく必要があります AWS IoT Events。 AWS IoT Events およびその他の AWS のサービスが IAM と連携する方法の概要を把握するには、「IAM ユーザーガイド」のAWS 「IAM と連携する のサービス」を参照してください。

トピック

- AWS IoT Events アイデンティティベースのポリシー
- AWS IoT Events リソースベースのポリシー
- AWS IoT Events タグに基づく認可
- AWS IoT Events IAM ロール

AWS IoT Events アイデンティティベースのポリシー

IAM アイデンティティベースのポリシーでは、許可または拒否されたアクションとリソースを指定でき、さらにアクションが許可または拒否された条件を指定できます。 AWS IoT Events は、特定のアクション、リソース、および条件キーをサポートします。JSON ポリシーで使用するすべての要素については、「IAM ユーザーガイド」の「IAM JSON ポリシーエレメントのリファレンス」を参照してください。

アクション

IAM アイデンティティベースのポリシーの `Action` エレメントは、そのポリシーにより許可または拒否される特定のアクションについて説明します。ポリシーアクションの名前は通常、関連する AWS API オペレーションと同じです。このアクションは、関連付けられたオペレーションを実行するためのアクセス許可を付与するポリシーで使用されます。

のポリシーアクションは、アクションの前にプレフィックス AWS IoT Events を使用しますiotevents:。例えば、 API オペレーションを使用して AWS IoT Events AWS IoT Events `CreateInput`入力を作成するアクセス許可を付与するには、ポリシーに `iotevents:CreateInput`アクションを含めます。API オペレーションで AWS IoT Events `BatchPutMessage`入力を送信するアクセス許可を付与するには、ポリシーに `iotevents-data:BatchPutMessage`アクションを含めます。ポリシーステートメントには、 `Action`または `NotAction` element を含める必要があります。 は、このサービスで実行できるタスクを記述する独自のアクションのセット AWS IoT Events を定義します。

単一のステートメントに複数のアクションを指定するには次のようにコンマで区切ります。

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

ワイルドカード (*) を使用して複数アクションを指定できます。例えば、`Describe` という単語で始まるすべてのアクションを指定するには次のアクションを含めます。

```
"Action": "iotevents:Describe*"
```

AWS IoT Events アクションのリストを確認するには、「IAM ユーザーガイド」の「 で定義されるアクション AWS IoT Events」を参照してください。

リソース

Resource エレメントは、アクションが適用されるオブジェクトを指定します。ステートメントには、Resource または NotResource エレメントを含める必要があります。ARN を使用して、またはステートメントがすべてのリソースに適用されることを示すワイルドカード *を使用して、リソースを指定します。

AWS IoT Events ディテクターモデルリソースには次の ARN があります。

```
arn:${Partition}:iotevents:${Region}:${Account}:detectorModel/${detectorModelName}
```

ARN の形式の詳細については、「Amazon AWS リソースネーム (ARNs) を使用してリソースを識別する」を参照してください。

例えば、ステートメントで Foobar ディテクターモデルを指定するには、次の ARN を使用します。

```
"Resource": "arn:aws:iotevents:us-east-1:123456789012:detectorModel/Foobar"
```

特定のアカウントに属するすべてのインスタンスを指定するには、ワイルドカード (*) を使用します。

```
"Resource": "arn:aws:iotevents:us-east-1:123456789012:detectorModel/*"
```

リソースを作成するためのアクションなど、一部の AWS IoT Events アクションは、特定のリソースで実行できません。このような場合はワイルドカード *を使用する必要があります。

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

一部の AWS IoT Events API アクションには、複数のリソースが含まれます。例えば CreateDetectorModel は、条件ステートメント内の入力を参照するため、ユーザーには入力およびディテクターモデルを使用する権限が必要です。複数リソースを単一ステートメントで指定するには、ARN をカンマで区切ります。

```
"Resource": [
      "resource1",
      "resource2"
```

AWS IoT Events リソースタイプとその ARNs「 で定義されるリソース AWS IoT Events」を参照してください。 どのアクションで各リソースの ARN を指定できるかについては、AWS IoT Eventsで定義されるアクションを参照してください。

条件キー

`Condition` エレメント またはCondition ブロックを使用すると、ステートメントが有効な条件を指定できます。`Condition` エレメントはオプションです。イコールや以下などの条件演算子を使用する条件表現を構築して、リクエスト内に値のあるポリシーの条件に一致させることができます。

1 つのステートメントに複数の `Condition` エレメントを指定する場合、または 1 つの `Condition` エレメントに複数のキーを指定する場合、 AWS が論理 AND 演算を使用してそれらを評価します。1 つの条件キーに複数の値を指定すると、 は論理ORオペレーションを使用して条件 AWS を評価します。ステートメントの権限が付与される前にすべての条件が満たされる必要があります。

条件を指定する際にプレースホルダー変数も使用できます。例えば、ユーザー名でタグ付けされている場合のみ、リソースにアクセスするユーザーアクセス許可を付与できます。詳細については、「IAM ユーザーガイド」の「IAM ポリシーエレメント。可変およびタグ」を参照してください。

AWS IoT Events はサービス固有の条件キーを提供しませんが、一部のグローバル条件キーの使用をサポートしています。すべての AWS グローバル条件キーを確認するには、「IAM ユーザーガイド」のAWS 「 グローバル条件コンテキストキー」を参照してください。

例

AWS IoT Events アイデンティティベースのポリシーの例を表示するには、「」を参照してくださいAWS IoT Events アイデンティティベースのポリシーの例。

AWS IoT Events リソースベースのポリシー

AWS IoT Events はリソースベースのポリシーをサポートしていません。」 詳細なリソースベースのポリシーページの例を表示するには、https://docs.aws.amazon.com/lambda/latest/dg/access-control-resource-based.html を参照してください。

AWS IoT Events タグに基づく認可

AWS IoT Events リソースにタグをアタッチしたり、リクエストでタグを渡すことができます AWS IoT Events。タグに基づいてアクセスを制御するには`iotevents:ResourceTag/`*`key-name`*、`aws:RequestTag/`*`key-name`*、または `aws:TagKeys` の条件キーを使用して、ポリシーの条件要素でタグ情報を提供します。 AWS IoT Events リソースのタグ付けの詳細については、「AWS IoT Events リソースのタグ付け」を参照してください。

リソースのタグに基づいてリソースへのアクセスを制限するためのアイデンティティベースのポリシーの例を表示するには、「タグに基づいて入力を表示する AWS IoT Events」を参照してください。

AWS IoT Events IAM ロール

IAM ロールは、特定のアクセス許可 AWS アカウント を持つ 内のエンティティです。

での一時的な認証情報の使用 AWS IoT Events

一時的な認証情報を使用して、フェデレーションでサインインする、IAM 役割を引き受ける、またはクロスアカウント役割を引き受けることができます。一時的なセキュリティ認証情報を取得するには、AssumeRole や GetFederationToken などの AWS Security Token Service（AWS STS) API オペレーションを呼び出します。

AWS IoT Events は、一時的な認証情報の使用をサポートしていません。

サービスにリンクされた役割

サービスにリンクされたロールを使用すると、 AWS サービスは他の サービスのリソースにアクセスして、ユーザーに代わってアクションを実行できます。サービスリンクロールは IAM アカウント内に表示され、サービスによって所有されます。IAM 管理者は、サービスリンクロールの許可を表示できますが、編集することはできません。

AWS IoT Events は、サービスにリンクされたロールをサポートしていません。

サービス役割

この機能により、ユーザーに代わってサービスがサービス役割を引き受けることが許可されます。この役割により、サービスがお客様に代わって他のサービスのリソースにアクセスし、アクションを完了することが許可されます。サービス役割はIAM アカウントに表示され、アカウントによって所有されます。つまり、IAM 管理者はこの役割の権限を変更できます。ただし、それにより、サービスの機能が損なわれる場合があります。

AWS IoT Events はサービスロールをサポートします。

AWS IoT Events アイデンティティベースのポリシーの例

デフォルトでは、ユーザーとロールには AWS IoT Events リソースを作成または変更するアクセス許可はありません。また、、 AWS Management Console AWS CLI、または AWS API を使用してタスクを実行することはできません。IAM 管理者は、ユーザーとロールに必要な、指定されたリソースで特定の API オペレーションを実行する権限をユーザーとロールに付与する IAM ポリシーを作成す

る必要があります。続いて、管理者はそれらのアクセス許可が必要なユーザーまたはグループにそのポリシーをアタッチします。

これらの JSON ポリシードキュメント例を使用して IAM のアイデンティティベースのポリシーを作成する方法については、『IAM ユーザーガイド』の「JSON タブでのポリシーの作成」を参照してください。

トピック

- ポリシーに関するベストプラクティス
- AWS IoT Events コンソールを使用する
- ユーザーが で自分のアクセス許可を表示できるようにする AWS IoT Events
- 1 つの AWS IoT Events 入力にアクセスする
- タグに基づいて入力を表示する AWS IoT Events

ポリシーに関するベストプラクティス

アイデンティティベースポリシーは非常に強力です。アカウント内の AWS IoT Events リソースを作成、アクセス、または削除できるかどうかを決定します。これらのアクションを実行すると、 AWS アカウントに料金が発生する可能性があります。アイデンティティベースポリシーを作成したり編集したりする際には、以下のガイドラインと推奨事項に従ってください:

- AWS 管理ポリシーを使用して開始する – の使用 AWS IoT Events をすばやく開始するには、AWS 管理ポリシーを使用して、従業員に必要なアクセス許可を付与します。これらのポリシーはアカウントで既に有効になっており、 AWSによって管理および更新されています。詳細については、「IAM ユーザーガイド」の「 AWS マネージドポリシーでアクセス許可の使用を開始する」を参照してください。
- 最小特権を付与する - カスタムポリシーを作成するときは、タスクを実行するために必要なアクセス許可のみを付与します。最小限の許可からスタートし、必要に応じて追加の許可を付与します。この方法は、寛容過ぎる許可から始めて、後から厳しくしようとするよりも安全です。詳細については、IAM ユーザーガイドの「最小特権を認める」を参照してください。
- 機密性の高いオペレーションに MFA を有効にする - セキュリティを強化するには、ユーザーが多要素認証 (MFA) を使用して機密性の高リソースまたは API オペレーションにアクセスすることを義務付けます。詳細については、IAM ユーザーガイドの「AWSでの多要素認証 (MFA) の使用」を参照してください。
- 追加セキュリティに対するポリシー条件を使用する - 実行可能な範囲内で、アイデンティティベースのポリシーがリソースにアクセスできる条件を定義します。例えば、あるリクエストの送信が許

可される IP アドレスの範囲を指定するための条件を記述できます。指定された日付または時間範囲内でのみリクエストを許可する条件を書くことも、SSL や MFA の使用を要求することもできます。詳細については、IAM ユーザーガイド」の「IAM JSON ポリシー要素: 条件」を参照してください。

AWS IoT Events コンソールを使用する

AWS IoT Events コンソールにアクセスするには、一連の最小限のアクセス許可が必要です。これらのアクセス許可により、 の AWS IoT Events リソースの詳細を一覧表示および表示できます AWS アカウント。最小限必要な許可よりも制限が厳しいアイデンティティベースのポリシーを作成すると、そのポリシーを持つエンティティ (ユーザーまたはロール) に対してコンソールが意図したとおりに機能しません。

これらのエンティティが引き続き AWS IoT Events コンソールを使用できるようにするには、エンティティに次の AWS 管理ポリシーもアタッチします。詳細については、IAM ユーザーガイド」の「ユーザーへの許可の追加」を参照してください。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iotevents-data:BatchPutMessage",
                "iotevents-data:BatchUpdateDetector",
                "iotevents:CreateDetectorModel",
                "iotevents:CreateInput",
                "iotevents:DeleteDetectorModel",
                "iotevents:DeleteInput",
                "iotevents-data:DescribeDetector",
                "iotevents:DescribeDetectorModel",
                "iotevents:DescribeInput",
                "iotevents:DescribeLoggingOptions",
                "iotevents:ListDetectorModelVersions",
                "iotevents:ListDetectorModels",
                "iotevents-data:ListDetectors",
                "iotevents:ListInputs",
                "iotevents:ListTagsForResource",
                "iotevents:PutLoggingOptions",
                "iotevents:TagResource",
                "iotevents:UntagResource",
```

```
                "iotevents:UpdateDetectorModel",
                "iotevents:UpdateInput",
                "iotevents:UpdateInputRouting"
            ],
            "Resource": "arn:${Partition}:iotevents:${Region}:${Account}:detectorModel/
${detectorModelName}",
            "Resource": "arn:${Partition}:iotevents:${Region}:${Account}:input/
${inputName}"
        }
    ]
}
```

AWS CLI または AWS API のみを呼び出すユーザーには、最小限のコンソールアクセス許可を付与する必要はありません。代わりに、実行しようとしている API オペレーションに一致するアクションのみへのアクセスが許可されます。

ユーザーが で自分のアクセス許可を表示できるようにする AWS IoT Events

この例では、ユーザー ID にアタッチされたインラインおよび管理ポリシーの表示を ユーザーに許可するポリシーを作成する方法を示します。ユーザーに自分の IAM アクセス許可の表示を許可すると、セキュリティ意識向上とセルフサービス機能に役立ちます。このポリシーには、コンソールで、または AWS CLI または AWS API を使用してプログラムでこのアクションを実行するアクセス許可が含まれています。

```
{
       "Version": "2012-10-17",
       "Statement": [
           {
               "Sid": "ViewOwnUserInfo",
               "Effect": "Allow",
               "Action": [
                   "iam:GetUserPolicy",
                   "iam:ListGroupsForUser",
                   "iam:ListAttachedUserPolicies",
                   "iam:ListUserPolicies",
                   "iam:GetUser"
               ],
               "Resource": [
                   "arn:aws:iam::*:user/${aws:username}"
               ]
           },
           {
```

```
                "Sid": "NavigateInConsole",
                "Effect": "Allow",
                "Action": [
                    "iam:GetGroupPolicy",
                    "iam:GetPolicyVersion",
                    "iam:GetPolicy",
                    "iam:ListAttachedGroupPolicies",
                    "iam:ListGroupPolicies",
                    "iam:ListPolicyVersions",
                    "iam:ListPolicies",
                    "iam:ListUsers"
                ],
                "Resource": "*"
            }
        ]
    }
```

1 つの AWS IoT Events 入力にアクセスする

AWS IoT Events 入力へのきめ細かなアクセスコントロールは、マルチユーザー環境またはマルチチーム環境でセキュリティを維持する上で重要です。このセクションでは、特定の AWS IoT Events 入力へのアクセスを許可し、他のユーザーへのアクセスを制限する IAM ポリシーを作成する方法を示します。

この例では、 AWS IoT Events 入力の 1 つである AWS アカウント へのアクセス権を のユーザーに付与できます`exampleInput`。また、入力の追加、更新、削除をユーザーに許可することもできます。

このポリシーは、`iotevents:ListInputs`、`iotevents:DescribeInput`、`iotevents:CreateInput`、`iotevent`アクセス許可をユーザーに付与します。ユーザーにアクセス許可を付与し、コンソールを使用してユーザーをテストする Amazon Simple Storage Service (Amazon S3) のチュートリアルの例については、「ユーザーポリシーによるバケットへのアクセスの制御」を参照してください。

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Sid":"ListInputsInConsole",
         "Effect":"Allow",
         "Action":[
            "iotevents:ListInputs"
```

```
            ],
            "Resource":"arn:aws:iotevents:::*"
        },
        {
            "Sid":"ViewSpecificInputInfo",
            "Effect":"Allow",
            "Action":[
                "iotevents:DescribeInput"
            ],
            "Resource":"arn:aws:iotevents:::exampleInput"
        },
        {
            "Sid":"ManageInputs",
            "Effect":"Allow",
            "Action":[
                "iotevents:CreateInput",
                "iotevents:DeleteInput",
                "iotevents:DescribeInput",
                "iotevents:ListInputs",
                "iotevents:UpdateInput"
            ],
            "Resource":"arn:aws:iotevents:::exampleInput/*"
        }
    ]
}
```

タグに基づいて入力を表示する AWS IoT Events

タグはリソースの整理に役立ちます AWS IoT Events 。アイデンティティベースのポリシーの条件を使用して、タグに基づいて AWS IoT Events リソースへのアクセスを制御できます。この例は、##の表示を許可するポリシーを作成する方法を示しています。ただし、許可は、##タグ Owner がそのユーザーのユーザー名の値を持っている場合にのみ付与されます。このポリシーでは、このアクションをコンソールで実行するために必要なアクセス許可も付与します。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ListInputsInConsole",
            "Effect": "Allow",
            "Action": "iotevents:ListInputs",
            "Resource": "*"
        },
```

```
        {
            "Sid": "ViewInputsIfOwner",
            "Effect": "Allow",
            "Action": "iotevents:ListInputs",
            "Resource": "arn:aws:iotevents:*:*:input/*",
            "Condition": {
                "StringEquals": {"aws:ResourceTag/Owner": "${aws:username}"}
            }
        }
    ]
}
```

このポリシーをアカウントの ユーザーにアタッチできます。という名前のユーザーが AWS IoT Events ##を表示richard-roeしようとする場合は、##に Owner=richard-roeまたは のタグを付ける必要がありますowner=richard-roe。それ以外の場合、アクセスは拒否されます。条件キー名では大文字と小文字が区別されないため、条件タグキー Owner は Owner と owner の両方に一致します。詳細については、「IAM ユーザーガイド」の「IAM JSON ポリシー要素: 条件」を参照してください。

のサービス間の混乱した代理の防止 AWS IoT Events

> **Note**
>
> - この AWS IoT Events サービスでは、リソースが作成されたのと同じアカウントでアクションを開始するためにのみ、ロールを使用できます。これにより、混乱した代理攻撃を防ぐことができます AWS IoT Events。
> - このページは、混乱した代理問題がどのように機能するかを確認するためのリファレンスとして機能し、クロスアカウントリソースがサービスで AWS IoT Events 許可された場合に防止できます。

混乱した代理問題は、アクションを実行するアクセス許可を持たないエンティティが、より権限のあるエンティティにアクションの実行を強制できるセキュリティ上の問題です。では AWS、サービス間のなりすましにより、混乱した代理問題が発生する可能性があります。

サービス間でのなりすましは、1 つのサービス (呼び出し元サービス) が、別のサービス (呼び出し対象サービス) を呼び出すときに発生する可能性があります。呼び出し元サービスは、本来ならアクセスすることが許可されるべきではない方法でその許可を使用して、別のお客様のリソースに対する処理を実行するように操作される場合があります。これを防ぐために、 は、アカウント内のリソース

へのアクセス権が付与されたサービスプリンシパルを使用して、すべてのサービスのデータを保護するのに役立つツール AWS を提供します。

リソースポリシーで `aws:SourceArn`および `aws:SourceAccount` グローバル条件コンテキストキーを使用して、 が別のサービス AWS IoT Events に付与するアクセス許可をリソースに制限することをお勧めします。`aws:SourceArn` の値に Amazon S3 バケット ARN などのアカウント ID が含まれていない場合は、両方のグローバル条件コンテキストキーを使用して、アクセス許可を制限する必要があります。同じポリシーステートメントでこれらのグローバル条件コンテキストキーの両方を使用し、アカウント ID に`aws:SourceArn` の値が含まれていない場合、`aws:SourceAccount` 値と `aws:SourceArn` 値の中のアカウントには、同じアカウント ID を使用する必要があります。

クロスサービスのアクセスにリソースを 1 つだけ関連付けたい場合は、`aws:SourceArn` を使用します。そのアカウントのリソースをクロスサービスの使用に関連付けることを許可する場合は、`aws:SourceAccount` を使用します。`aws:SourceArn` の値は、`sts:AssumeRole`リクエストに関連付けられているディテクタモデルまたはアラームモデルである必要があります。

混乱した代理問題から保護するための最も効果的な方法は、リソースの完全な ARN で `aws:SourceArn` グローバル条件コンテキストキーを使用することです。リソースの完全な ARN が不明な場合や、複数のリソースを指定する場合は、`aws:SourceArn` グローバルコンテキスト条件キーを使用して、ARN の未知部分をワイルドカード (*) で表します。例えば、`arn:aws:`*`iotevents`*`:*:`*`123456789012`*`:*` と指定します。

次の例は、 で `aws:SourceArn`および `aws:SourceAccount` グローバル条件コンテキストキーを使用して、混乱した代理問題 AWS IoT Events を回避する方法を示しています。

トピック

- 例: AWS IoT Events ディテクターモデルへの安全なアクセス
- 例: AWS IoT Events アラームモデルへの安全なアクセス
- 例: 指定されたリージョンの AWS IoT Events リソースにアクセスする
- 例: のログ記録オプションを設定する AWS IoT Events

例: AWS IoT Events ディテクターモデルへの安全なアクセス

この例では、 で特定のディテクターモデルへのアクセスを安全に許可する IAM ポリシーを作成する方法を示します AWS IoT Events。このポリシーでは、条件を使用して、指定された AWS アカウントと AWS IoT Events サービスのみがロールを引き受けることができるようにし、セキュリティレイヤーを追加します。この例では、ロールは *`WindTurbine01`* という名前のディテクターモデルにのみアクセスできます。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": [
          "iotevents.amazonaws.com"
        ]
      },
      "Action": "sts:AssumeRole",
      "Condition": {
        "StringEquals": {
          "aws:SourceAccount": "account_id"
        },
        "ArnEquals": {
          "aws:SourceArn":
 "arn:aws:iotevents:region:account_id:detectorModel/WindTurbine01"
        }
      }
    }
  ]
 }
```

例: AWS IoT Events アラームモデルへの安全なアクセス

この例では、 がアラームモデル AWS IoT Events に安全にアクセスできるようにする IAM ポリシーを作成する方法を示します。このポリシーは、条件を使用して、指定された AWS アカウントと AWS IoT Events サービスのみがロールを引き受けることができるようにします。

この例では、ロールは、アラームモデル ARN の*ワイルドカードで示されるように、指定された AWS アカウント内の任意のアラームモデルにアクセスできます。`aws:SourceAccount` および `aws:SourceArn`条件は、混乱した代理問題を防ぐために連携します。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": [
          "iotevents.amazonaws.com"
```

```
        ]
      },
      "Action": "sts:AssumeRole",
      "Condition": {
        "StringEquals": {
          "aws:SourceAccount": "account_id"
        },
        "ArnEquals": {
          "aws:SourceArn": "arn:aws:iotevents:region:account_id:alarmModel/*"
        }
      }
    }
  ]
}
```

例: 指定されたリージョンの AWS IoT Events リソースにアクセスする

この例では、特定の AWS リージョンの AWS IoT Events リソースにアクセスするように IAM ロールを設定する方法を示します。IAM ポリシーでリージョン固有の ARNs を使用することで、さまざまな地理的エリアの AWS IoT Events リソースへのアクセスを制限できます。このアプローチは、マルチリージョンデプロイでセキュリティとコンプライアンスを維持するのに役立ちます。この例のリージョンは *us-east-1* です。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": [
          "iotevents.amazonaws.com"
        ]
      },
      "Action": "sts:AssumeRole",
      "Condition": {
        "StringEquals": {
          "aws:SourceAccount": "account_id"
        },
        "ArnEquals": {
          "aws:SourceArn": "arn:aws:iotevents:us-east-1:account_id:*"
        }
      }
    }
```

```
  ]
}
```

例: のログ記録オプションを設定する AWS IoT Events

アプリケーションのモニタリング、デバッグ、監査 AWS IoT Events には、適切なログ記録が重要です。このセクションでは、 で使用できるログ記録オプションの概要を説明します AWS IoT Events。

この例では、 が CloudWatch Logs AWS IoT Events にデータをログ記録できるようにする IAM ロールを設定する方法を示します。リソース ARN でワイルドカード (*) を使用すると、 AWS IoT Events インフラストラクチャ全体で包括的なログ記録が可能になります。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": [
          "iotevents.amazonaws.com"
        ]
      },
      "Action": "sts:AssumeRole",
      "Condition": {
        "StringEquals": {
          "aws:SourceAccount": "account_id"
        },
        "ArnEquals": {
          "aws:SourceArn": "arn:aws:iotevents:region:account_id:*"
        }
      }
    }
  ]
}
```

AWS IoT Events ID とアクセスのトラブルシューティング

次の情報は、 と IAM の使用時に発生する可能性がある一般的な問題の診断 AWS IoT Events と修正に役立ちます。

トピック

- でアクションを実行する権限がない AWS IoT Events
- iam:PassRole を実行する権限がない
- 自分の 以外のユーザーに自分の AWS IoT Events リソース AWS アカウント へのアクセスを許可したい

でアクションを実行する権限がない AWS IoT Events

からアクションを実行する権限がないと AWS Management Console 通知された場合は、管理者に連絡してサポートを依頼する必要があります。担当の管理者はお客様のユーザー名とパスワードを発行した人です。

次のエラー例は、`mateojackson` IAM ユーザーがコンソールを使用して*##*に関する詳細を表示しようとしたが、`iotevents:`*`ListInputs`* 許可を持っていない場合に発生します。

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 iotevents:ListInputs on resource: my-example-input
```

この場合、Mateo は管理者に、`iotevents:`*`ListInput`* アクションを使用して *`my-example-input`* リソースにアクセスできるようにポリシーを更新するように依頼します。

`iam:PassRole` を実行する権限がない

`iam:PassRole` アクションを実行する権限がないというエラーが表示された場合は、ポリシーを更新して AWS IoT Eventsにロールを渡すことができるようにする必要があります。

一部の AWS のサービス では、新しいサービスロールまたはサービスにリンクされたロールを作成する代わりに、既存のロールをそのサービスに渡すことができます。そのためには、サービスにロールを渡す権限が必要です。

以下の例のエラーは、`marymajor` という IAM ユーザーがコンソールを使用して AWS IoT Eventsでアクションを実行しようする場合に発生します。ただし、このアクションをサービスが実行するには、サービスロールから付与された権限が必要です。メアリーには、ロールをサービスに渡す許可がありません。

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

この場合、Mary のポリシーを更新してメアリーに `iam:PassRole` アクションの実行を許可する必要があります。

サポートが必要な場合は、 AWS 管理者にお問い合わせください。サインイン認証情報を提供した担当者が管理者です。

自分の 以外のユーザーに自分の AWS IoT Events リソース AWS アカウント へのアクセスを許可したい

他のアカウントのユーザーや組織外の人が、リソースにアクセスするために使用できるロールを作成できます。ロールの引き受けを委託するユーザーを指定できます。リソースベースのポリシーまたはアクセスコントロールリスト (ACL) をサポートするサービスの場合、それらのポリシーを使用して、リソースへのアクセスを付与できます。

最適なオプションについては、以下のトピックを参照してください。

- がこれらの機能 AWS IoT Events をサポートしているかどうかを確認するには、「」を参照してくださいと IAM の AWS IoT Events 連携方法。
- 所有 AWS アカウント している 全体のリソースへのアクセスを提供する方法については、IAM ユーザーガイドの「所有 AWS アカウント している別の の IAM ユーザーへのアクセスを提供する」を参照してください。
- リソースへのアクセスをサードパーティーに提供する方法については AWS アカウント、「IAM ユーザーガイド」の「サードパーティー AWS アカウント が所有する へのアクセスを提供する」を参照してください。
- ID フェデレーションを介してアクセスを提供する方法については、「IAM ユーザーガイド」の「外部で認証されたユーザー (ID フェデレーション) へのアクセスの許可」を参照してください。
- クロスアカウントアクセスにおけるロールとリソースベースのポリシーの使用方法の違いについては、「IAM ユーザーガイド」の「IAM でのクロスアカウントのリソースへのアクセス」を参照してください。

信頼性、可用性、パフォーマンス AWS IoT Events を維持するためのモニタリング

モニタリングは、 および AWS ソリューションの信頼性、可用性、パフォーマンスを維持する上で重要な部分 AWS IoT Events です。マルチポイント障害が発生した場合は、その障害をより簡単にデバッグできるように、 AWS ソリューションのすべての部分からモニタリングデータを収集する必要があります。モニタリングを開始する前に AWS IoT Events、以下の質問に対する回答を含むモニタリング計画を作成する必要があります。

- どのような目的でモニタリングしますか?

- どのリソースをモニタリングしますか?
- どのくらいの頻度でこれらのリソースをモニタリングしますか?
- どのモニタリングツールを使用しますか?
- 誰がモニタリングタスクを実行しますか?
- 問題が発生したときに誰が通知を受け取りますか?

次のステップでは、さまざまなタイミングと負荷条件で AWS IoT Events パフォーマンスを測定することで、環境で通常のパフォーマンスのベースラインを確立します。 AWS IoT Eventsをモニタリングするときは、履歴モニタリングデータを保存して、現在のパフォーマンスデータと比較し、通常のパフォーマンスパターンとパフォーマンスの異常を特定し、問題に対処する方法を考案できるようにします。

例えば、Amazon EC2 を使用している場合は、インスタンスの CPU 使用率、ディスク I/O、ネットワーク使用率をモニタリングできます。確立したベースラインからパフォーマンスが外れた場合は、インスタンスの再設定または最適化を行って CPU 使用率の抑制、ディスク I/O の改善、またはネットワークトラフィックの低減を行うことが必要な場合があります。

トピック

- モニタリングできるツール AWS IoT Events
- Amazon CloudWatch AWS IoT Events によるモニタリング
- を使用した AWS IoT Events API コールのログ記録 AWS CloudTrail

モニタリングできるツール AWS IoT Events

AWS には、モニタリングに使用できるさまざまなツールが用意されています AWS IoT Events。これらのツールの一部はモニタリングを行うように設定できますが、一部のツールは手動による介入が必要です。モニタリングタスクをできるだけ自動化することをお勧めします。

自動モニタリングツール

次の自動モニタリングツールを使用して、問題を監視 AWS IoT Events して報告できます。

- Amazon CloudWatch Logs – AWS CloudTrail またはその他のソースからのログファイルをモニタリング、保存、およびアクセスします。詳細については、「Amazon CloudWatch ユーザーガイド」の「Amazon CloudWatch ダッシュボードの使用」を参照してください。

- AWS CloudTrail ログモニタリング – アカウント間でログファイルを共有し、CloudWatch Logs に送信CloudWatch CloudTrail ログファイルをリアルタイムでモニタリングし、Java でログ処理アプリケーションを書き込み、CloudTrail による配信後にログファイルが変更されていないことを確認します。詳細については、AWS CloudTrail ユーザーガイドのCloudTrail ログファイルの操作を参照してください。

手動モニタリングツール

モニタリングのもう 1 つの重要な点は AWS IoT Events 、CloudWatch アラームでカバーされていない項目を手動でモニタリングすることです。 AWS IoT Events、CloudWatch、およびその他の AWS コンソールダッシュボードには、 AWS 環境の状態がat-a-glanceビューが表示されます。ログファイルも確認することをお勧めします AWS IoT Events。

- AWS IoT Events コンソールには以下が表示されます。
  - デテクターモデル
  - デテクター
  - 入力
  - 設定
- CloudWatch のホームページには、以下の情報が表示されます。
  - 現在のアラームとステータス
  - アラームとリソースのグラフ
  - サービスのヘルスステータス

  また、CloudWatch を使用して、次のことが可能です。
  - 作成 関心のあるサービスをモニタリングする CloudWatch ダッシュボードの作成
  - メトリクスデータをグラフ化して、問題のトラブルシューティングを行い、傾向を確認する
  - すべての AWS リソースメトリクスを検索して参照する
  - 問題があることを通知するアラームを作成/編集する

Amazon CloudWatch AWS IoT Events によるモニタリング

AWS IoT Events ディテクターモデルを開発またはデバッグするとき AWS IoT Events は、 が何をしているのか、および発生しているエラーを知る必要があります。Amazon CloudWatch は、 AWS リソースと AWS で実行されるアプリケーションをリアルタイムでモニタリングします。CloudWatch を使用すると、リソースの使用、アプリケーションのパフォーマンス、運用ヘルスをシステム全体

で把握できます。 AWS IoT Events ディテクターモデルの開発時に Amazon CloudWatch ログ記録を有効にする には、 AWS IoT Eventsの CloudWatch ロギングを有効にする方法に関する情報があります。以下に示すようなログを生成するには、[Level of verbosity] (詳細レベル) を Debug (デバッグ) に設定し、[Detector Model Name] (ディテクターモデル名) およびオプションの [KeyValue] (キー値) である [Debug Targets] (デバッグターゲット) を 1 つ以上指定する必要があります。

次の例は、 AWS IoT Eventsによって生成された CloudWatch DEBUG レベルのログエントリを示しています。

```
{
  "timestamp": "2019-03-15T15:56:29.412Z",
  "level": "DEBUG",
  "logMessage": "Summary of message evaluation",
  "context": "MessageEvaluation",
  "status": "Success",
  "messageId": "SensorAggregate_2th846h",
  "keyValue": "boiler_1",
  "detectorModelName": "BoilerAlarmDetector",
  "initialState": "high_temp_alarm",
  "initialVariables": {
    "high_temp_count": 1,
    "high_pressure_count": 1
  },
  "finalState": "no_alarm",
  "finalVariables": {
    "high_temp_count": 0,
    "high_pressure_count": 0
  },
  "message": "{ \"temp\": 34.9, \"pressure\": 84.5}",
  "messageType": "CUSTOMER_MESSAGE",
  "conditionEvaluationResults": [
    {
      "result": "True",
      "eventName": "alarm_cleared",
      "state": "high_temp_alarm",
      "lifeCycle": "OnInput",
      "hasTransition": true
    },
    {
      "result": "Skipped",
      "eventName": "alarm_escalated",
      "state": "high_temp_alarm",
      "lifeCycle": "OnInput",
```

```
      "hasTransition": true,
      "resultDetails": "Skipped due to transition from alarm_cleared event"
    },
    {
      "result": "True",
      "eventName": "should_recall_technician",
      "state": "no_alarm",
      "lifeCycle": "OnEnter",
      "hasTransition": true
    }
  ]
}
```

を使用した AWS IoT Events API コールのログ記録 AWS CloudTrail

AWS IoT Events は、ユーザー AWS CloudTrail、ロール、または のサービスによって実行されたアクションを記録する AWS サービスである と統合されています AWS IoT Events。CloudTrail は、AWS IoT Events コンソールからの呼び出しや API へのコード呼び出しを含む、 のすべての API コールをイベント AWS IoT Events としてキャプチャします。 AWS IoT Events APIs

証跡を作成する場合は、 イベントなど、Amazon S3 バケットへの CloudTrail イベントの継続的な配信を有効にすることができます AWS IoT Events。証跡を設定しない場合でも、CloudTrail コンソールの [イベント履歴] で最新のイベントを表示できます。CloudTrail で収集された情報を使用して、リクエストの実行元の IP アドレス AWS IoT Events、リクエストの実行者、リクエストの実行日時などの詳細を確認できます。

CloudTrail の詳細については、「AWS CloudTrail ユーザーガイド」を参照してください。

AWS IoT Events CloudTrail の情報

CloudTrail は、 AWS アカウントの作成時にアカウントで有効になります。アクティビティが発生すると AWS IoT Events、そのアクティビティは CloudTrail イベントに記録され、イベント履歴に他の AWS サービスイベントが記録されます。 AWS アカウントで最近のイベントを表示、検索、ダウンロードできます。詳細については、「CloudTrail イベント履歴の操作」を参照してください。

のイベントなど、 AWS アカウントのイベントの継続的な記録については AWS IoT Events、証跡を作成します。追跡により、CloudTrail はログファイルを Amazon S3 バケットに配信できます。デフォルトでは、コンソールで証跡を作成すると、証跡はすべての AWS リージョンに適用されます。証跡は、 AWS パーティション内のすべてのリージョンからのイベントをログに記録し、指定した Amazon S3 バケットにログファイルを配信します。さらに、CloudTrail ログで収集されたイベント

データをさらに分析して処理するように、他の AWS サービスを設定できます。詳細については、以下を参照してください。

- AWS アカウントの証跡の作成
- CloudTrail のサポートされているサービスと統合
- 「CloudTrail の Amazon SNS 通知の設定」
- 「複数のリージョンから CloudTrail ログファイルを受け取る」および「複数のアカウントから CloudTrail ログファイルを受け取る」

各イベントまたはログエントリには、誰がリクエストを生成したかという情報が含まれます。同一性情報は次の判断に役立ちます。

- リクエストが、ルートと IAM ユーザー認証情報のどちらを使用して送信されたか。
- リクエストがロールまたはフェデレーションユーザーの一時的なセキュリティ認証情報を使用して行われたかどうか。
- リクエストが別の AWS サービスによって行われたかどうか。

詳細については、AWS IoT Events 「 API リファレンス」のCloudTrail userIdentity element. AWS IoT Events actions」を参照してください。

AWS IoT Events ログファイルエントリについて

証跡は、指定した Amazon S3 バケットにイベントをログファイルとして配信できるようにする設定です。 AWS CloudTrail ログファイルには 1 つ以上のログエントリが含まれます。イベントは、任意のリソースからの単一の要求を表し、要求されたアクション、アクションの日時、要求パラメータなどに関する情報が含まれます。CloudTrail ログファイルは、公開 API コールの順序付けられたスタックトレースではないため、特定の順序では表示されません。

AWS アカウントで CloudTrail ログ記録が有効になっている場合、 AWS IoT Events アクションに対して行われたほとんどの API コールは CloudTrail ログファイルに記録され、他の AWS サービスレコードとともに書き込まれます。CloudTrail は、期間とファイルサイズに基づいて、新しいファイルをいつ作成して書き込むかを決定します。

各ログエントリには、リクエストの生成者に関する情報が含まれます。ログエントリのユーザーアイデンティティ情報は、次のことを確認するのに役立ちます。

- リクエストがルートまたは IAM ユーザー認証情報で行われたかどうか。

- リクエストがロールまたはフェデレーションユーザーの一時的なセキュリティ認証情報を使用して行われたかどうか。
- リクエストが別の AWS サービスによって行われたかどうか。

必要な場合はログファイルを自身の Amazon S3 バケットに保存できますが、ログファイルを自動的にアーカイブまたは削除するように Amazon S3 ライフサイクルルールを定義することもできます。デフォルトでは Amazon S3 のサーバー側の暗号化 (SSE) を使用して、ログファイルが暗号化されます。

ログファイルの配信時に通知を受け取りたい場合は、新しいログファイルの配信時に Amazon SNS 通知が発行されるように CloudTrail を設定できます。詳細については、CloudTrail 用の Amazon SNS 通知の設定を参照してください。

複数の AWS リージョンと複数の AWS アカウントの AWS IoT Events ログファイルを 1 つの Amazon S3 バケットに集約することもできます。

詳細は、CloudTrail ログ ファイルを複数のリージョンから受け取るとと複数のアカウントから CloudTrail ログファイルを受け取るを参照してください。

例: CloudTrail の DescribeDetector アクション

以下の例は、`DescribeDetector` アクションを示す CloudTrail ログエントリです。

```
{
  "eventVersion": "1.05",
  "userIdentity": {
    "type": "AssumedRole",
    "principalId": "AKIAI44QH8DHBEXAMPLE",
    "arn": "arn:aws:sts::123456789012:assumed-role/Admin/bertholt-brecht",
    "accountId": "123456789012",
    "accessKeyId": "access-key-id",
    "sessionContext": {
      "attributes": {
        "mfaAuthenticated": "false",
        "creationDate": "2019-02-08T18:53:58Z"
      },
      "sessionIssuer": {
        "type": "Role",
        "principalId": "AKIAI44QH8DHBEXAMPLE",
        "arn": "arn:aws:iam::123456789012:role/Admin",
        "accountId": "123456789012",
```

```
        "userName": "Admin"
      }
    }
  },
  "eventTime": "2019-02-08T19:02:44Z",
  "eventSource": "iotevents.amazonaws.com",
  "eventName": "DescribeDetector",
  "awsRegion": "us-east-1",
  "sourceIPAddress": "192.168.0.1",
  "userAgent": "aws-cli/1.15.65 Python/3.7.1 Darwin/16.7.0 botocore/1.10.65",
  "requestParameters": {
    "detectorModelName": "pressureThresholdEventDetector-brecht",
    "keyValue": "1"
  },
  "responseElements": null,
  "requestID": "00f41283-ea0f-4e85-959f-bee37454627a",
  "eventID": "5eb0180d-052b-49d9-a289-0eb8d08d4c27",
  "eventType": "AwsApiCall",
  "recipientAccountId": "123456789012"
}
```

例: CloudTrail の CreateDetectorModel アクション

以下の例は、`CreateDetectorModel` アクションを示す CloudTrail ログエントリです。

```
{
  "eventVersion": "1.05",
  "userIdentity": {
    "type": "AssumedRole",
    "principalId": "AKIAI44QH8DHBEXAMPLE:IotEvents-Lambda",
    "arn": "arn:aws:sts::123456789012:assumed-role/IotEvents-RoleForIotEvents-
ABC123DEF456/IotEvents-Lambda",
    "accountId": "123456789012",
    "accessKeyId": "AKIAI44QH8DHBEXAMPLE",
    "sessionContext": {
      "attributes": {
        "mfaAuthenticated": "false",
        "creationDate": "2019-02-07T22:22:30Z"
      },
      "sessionIssuer": {
        "type": "Role",
        "principalId": "AKIAI44QH8DHBEXAMPLE",
        "arn": "arn:aws:iam::123456789012:role/IotEventsLambda-RoleForIotEvents-
ABC123DEF456",
```

```
        "accountId": "123456789012",
        "userName": "IotEventsLambda-RoleForIotEvents-ABC123DEF456"
      }
    }
  },
  "eventTime": "2019-02-07T23:54:43Z",
  "eventSource": "iotevents.amazonaws.com",
  "eventName": "CreateDetectorModel",
  "awsRegion": "us-east-1",
  "sourceIPAddress": "192.168.0.1",
  "userAgent": "aws-internal/3",
  "requestParameters": {
    "detectorModelName": "myDetectorModel",
    "key": "HIDDEN_DUE_TO_SECURITY_REASONS",
    "roleArn": "arn:aws:iam::123456789012:role/events_action_execution_role"
  },
  "responseElements": null,
  "requestID": "cecfbfa1-e452-4fa6-b86b-89a89f392b66",
  "eventID": "8138d46b-50a3-4af0-9c5e-5af5ef75ea55",
  "eventType": "AwsApiCall",
  "recipientAccountId": "123456789012"
}
```

例: CloudTrail の CreateInput アクション

以下の例は、`CreateInput` アクションを示す CloudTrail ログエントリです。

```
{
  "eventVersion": "1.05",
  "userIdentity": {
    "type": "AssumedRole",
    "principalId": "AKIAI44QH8DHBEXAMPLE:IotEvents-Lambda",
    "arn": "arn:aws:sts::123456789012:assumed-role/IotEventsLambda-RoleForIotEvents-
ABC123DEF456/IotEvents-Lambda",
    "accountId": "123456789012",
    "accessKeyId": "AKIAI44QH8DHBEXAMPLE",
    "sessionContext": {
      "attributes": {
        "mfaAuthenticated": "false",
        "creationDate": "2019-02-07T22:22:30Z"
      },
      "sessionIssuer": {
        "type": "Role",
        "principalId": "AKIAI44QH8DHBEXAMPLE",
```

```
        "arn": "arn:aws:iam::123456789012:role/IotEventsLambda-RoleForIotEvents-
ABC123DEF456",
        "accountId": "123456789012",
        "userName": "IotEventsLambda-RoleForIotEvents-ABC123DEF456"
      }
    }
  },
  "eventTime": "2019-02-07T23:54:43Z",
  "eventSource": "iotevents.amazonaws.com",
  "eventName": "CreateInput",
  "awsRegion": "us-east-1",
  "sourceIPAddress": "192.168.0.1",
  "userAgent": "aws-internal/3",
  "requestParameters": {
    "inputName": "batchputmessagedetectorupdated",
    "inputDescription": "batchputmessagedetectorupdated"
  },
  "responseElements": null,
  "requestID": "fb315af4-39e9-4114-94d1-89c9183394c1",
  "eventID": "6d8cf67b-2a03-46e6-bbff-e113a7bded1e",
  "eventType": "AwsApiCall",
  "recipientAccountId": "123456789012"
}
```

例: CloudTrail の DeleteDetectorModel アクション

以下の例は、`DeleteDetectorModel` アクションを示す CloudTrail ログエントリです。

```
{
  "eventVersion": "1.05",
  "userIdentity": {
    "type": "AssumedRole",
    "principalId": "AKIAI44QH8DHBEXAMPLE:IotEvents-EventsLambda",
    "arn": "arn:aws:sts::123456789012:assumed-role/IotEventsLambda-RoleForIotEvents-
ABCD123DEF456/IotEvents-EventsLambda",
    "accountId": "123456789012",
    "accessKeyId": "AKIAI44QH8DHBEXAMPLE",
    "sessionContext": {
      "attributes": {
        "mfaAuthenticated": "false",
        "creationDate": "2019-02-07T22:22:30Z"
      },
      "sessionIssuer": {
        "type": "Role",
```

```
        "principalId": "AKIAI44QH8DHBEXAMPLE",
        "arn": "arn:aws:iam::123456789012:role/IotEventsLambda-RoleForIotEvents-
ABCD123DEF456",
        "accountId": "123456789012",
        "userName": "IotEventsLambda-RoleForIotEvents-ABCD123DEF456"
      }
    }
  },
  "eventTime": "2019-02-07T23:54:11Z",
  "eventSource": "iotevents.amazonaws.com",
  "eventName": "DeleteDetectorModel",
  "awsRegion": "us-east-1",
  "sourceIPAddress": "192.168.0.1",
  "userAgent": "aws-internal/3",
  "requestParameters": {
    "detectorModelName": "myDetectorModel"
  },
  "responseElements": null,
  "requestID": "149064c1-4e24-4160-a5b2-1065e63ee2e4",
  "eventID": "7669db89-dcc0-4c42-904b-f24b764dd808",
  "eventType": "AwsApiCall",
  "recipientAccountId": "123456789012"
}
```

例: CloudTrail の DeleteInput アクション

以下の例は、`DeleteInput` アクションを示す CloudTrail ログエントリです。

```
{
  "eventVersion": "1.05",
  "userIdentity": {
    "type": "AssumedRole",
    "principalId": "AKIAI44QH8DHBEXAMPLE:IotEvents-EventsLambda",
    "arn": "arn:aws:sts::123456789012:assumed-role/IotEventsLambda-RoleForIotEvents-
ABCD123DEF456/IotEvents-EventsLambda",
    "accountId": "123456789012",
    "accessKeyId": "AKIAI44QH8DHBEXAMPLE",
    "sessionContext": {
      "attributes": {
        "mfaAuthenticated": "false",
        "creationDate": "2019-02-07T22:22:30Z"
      },
      "sessionIssuer": {
        "type": "Role",
```

```
        "principalId": "AKIAI44QH8DHBEXAMPLE",
        "arn": "arn:aws:iam::123456789012:role/IotEventsLambda-RoleForIotEvents-
ABCD123DEF456",
        "accountId": "123456789012",
        "userName": "IotEventsLambda-RoleForIotEvents-ABCD123DEF456"
      }
    }
  },
  "eventTime": "2019-02-07T23:54:38Z",
  "eventSource": "iotevents.amazonaws.com",
  "eventName": "DeleteInput",
  "awsRegion": "us-east-1",
  "sourceIPAddress": "192.168.0.1",
  "userAgent": "aws-internal/3",
  "errorCode": "ResourceNotFoundException",
  "errorMessage": "Input of name: NoSuchInput not found",
  "requestParameters": {
    "inputName": "NoSuchInput"
  },
  "responseElements": null,
  "requestID": "ce6d28ac-5baf-423d-a5c3-afd009c967e3",
  "eventID": "be0ef01d-1c28-48cd-895e-c3ff3172c08e",
  "eventType": "AwsApiCall",
  "recipientAccountId": "123456789012"
}
```

例: CloudTrail の DescribeDetectorModel アクション

以下の例は、`DescribeDetectorModel` アクションを示す CloudTrail ログエントリです。

```
{
  "eventVersion": "1.05",
  "userIdentity": {
    "type": "AssumedRole",
    "principalId": "AKIAI44QH8DHBEXAMPLE:IotEvents-EventsLambda",
    "arn": "arn:aws:sts::123456789012:assumed-role/IotEventsLambda-RoleForIotEvents-
ABCD123DEF456/IotEvents-EventsLambda",
    "accountId": "123456789012",
    "accessKeyId": "AKIAI44QH8DHBEXAMPLE",
    "sessionContext": {
      "attributes": {
        "mfaAuthenticated": "false",
        "creationDate": "2019-02-07T22:22:30Z"
      },
```

```
      "sessionIssuer": {
        "type": "Role",
        "principalId": "AAKIAI44QH8DHBEXAMPLE",
        "arn": "arn:aws:iam::123456789012:role/IotEventsLambda-RoleForIotEvents-
ABCD123DEF456",
        "accountId": "123456789012",
        "userName": "IotEventsLambda-RoleForIotEvents-ABCD123DEF456"
      }
    }
  },
  "eventTime": "2019-02-07T23:54:20Z",
  "eventSource": "iotevents.amazonaws.com",
  "eventName": "DescribeDetectorModel",
  "awsRegion": "us-east-1",
  "sourceIPAddress": "192.168.0.1",
  "userAgent": "aws-internal/3",
  "requestParameters": {
    "detectorModelName": "myDetectorModel"
  },
  "responseElements": null,
  "requestID": "18a11622-8193-49a9-85cb-1fa6d3929394",
  "eventID": "1ad80ff8-3e2b-4073-ac38-9cb3385beb04",
  "eventType": "AwsApiCall",
  "recipientAccountId": "123456789012"
}
```

例: CloudTrail の DescribeInput アクション

以下の例は、`DescribeInput` アクションを示す CloudTrail ログエントリです。

```
{
  "eventVersion": "1.05",
  "userIdentity": {
    "type": "AssumedRole",
    "principalId": "AKIAI44QH8DHBEXAMPLE:IotEvents-EventsLambda",
    "arn": "arn:aws:sts::123456789012:assumed-role/IotEventsLambda-RoleForIotEvents-
ABCD123DEF456/IotEvents-EventsLambda",
    "accountId": "123456789012",
    "accessKeyId": "AAKIAI44QH8DHBEXAMPLE",
    "sessionContext": {

      "attributes": {
        "mfaAuthenticated": "false",
        "creationDate": "2019-02-07T22:22:30Z"
```

```
      },
      "sessionIssuer": {
        "type": "Role",
        "principalId": "AKIAI44QH8DHBEXAMPLE",
        "arn": "arn:aws:iam::123456789012:role/IotEventsLambda-RoleForIotEvents-
ABCD123DEF456",
        "accountId": "123456789012",
        "userName": "IotEventsLambda-RoleForIotEvents-ABCD123DEF456"
      }
    }
  },
  "eventTime": "2019-02-07T23:56:09Z",
  "eventSource": "iotevents.amazonaws.com",
  "eventName": "DescribeInput",
  "awsRegion": "us-east-1",
  "sourceIPAddress": "192.168.0.1",
  "userAgent": "aws-internal/3",
  "requestParameters": {
    "inputName": "input_createinput"
  },
  "responseElements": null,
  "requestID": "3af641fa-d8af-41c9-ba77-ac9c6260f8b8",
  "eventID": "bc4e6cc0-55f7-45c1-b597-ec99aa14c81a",
  "eventType": "AwsApiCall",
  "recipientAccountId": "123456789012"
}
```

例: CloudTrail の DescribeLoggingOptions アクション

以下の例は、`DescribeLoggingOptions` アクションを示す CloudTrail ログエントリです。

```
{
  "eventVersion": "1.05",
  "userIdentity": {
    "type": "AssumedRole",
    "principalId": "AKIAI44QH8DHBEXAMPLE:IotEvents-EventsLambda",
    "arn": "arn:aws:sts::123456789012:assumed-role/IotEventsLambda-RoleForIotEvents-
ABCD123DEF456/IotEvents-EventsLambda",
    "accountId": "123456789012",
    "accessKeyId": "AKIAI44QH8DHBEXAMPLE",
    "sessionContext": {
      "attributes": {
        "mfaAuthenticated": "false",
        "creationDate": "2019-02-07T22:22:30Z"
```

```
      },
      "sessionIssuer": {
        "type": "Role",
        "principalId": "AKIAI44QH8DHBEXAMPLE",
        "arn": "arn:aws:iam::123456789012:role/IotEventsLambda-RoleForIotEvents-
ABCD123DEF456",
        "accountId": "123456789012",
        "userName": "IotEventsLambda-RoleForIotEvents-ABCD123DEF456"
      }
    }
  },
  "eventTime": "2019-02-07T23:53:23Z",
  "eventSource": "iotevents.amazonaws.com",
  "eventName": "DescribeLoggingOptions",
  "awsRegion": "us-east-1",
  "sourceIPAddress": "192.168.0.1",
  "userAgent": "aws-internal/3",
  "requestParameters": null,
  "responseElements": null,
  "requestID": "b624b6c5-aa33-41d8-867b-025ec747ee8f",
  "eventID": "9c7ce626-25c8-413a-96e7-92b823d6c850",
  "eventType": "AwsApiCall",
  "recipientAccountId": "123456789012"
}
```

例: CloudTrail の ListDetectorModels アクション

以下の例は、`ListDetectorModels` アクションを示す CloudTrail ログエントリです。

```
{
  "eventVersion": "1.05",
  "userIdentity": {
    "type": "AssumedRole",
    "principalId": "AKIAI44QH8DHBEXAMPLE:IotEvents-EventsLambda",
    "arn": "arn:aws:sts::123456789012:assumed-role/IotEventsLambda-RoleForIotEvents-
ABCD123DEF456/IotEvents-EventsLambda",
    "accountId": "123456789012",
    "accessKeyId": "AKIAI44QH8DHBEXAMPLE",
    "sessionContext": {
      "attributes": {
        "mfaAuthenticated": "false",
        "creationDate": "2019-02-07T22:22:30Z"
      },
      "sessionIssuer": {
```

```
        "type": "Role",
        "principalId": "AKIAI44QH8DHBEXAMPLE",
        "arn": "arn:aws:iam::123456789012:role/IotEventsLambda-RoleForIotEvents-
ABCD123DEF456",
        "accountId": "123456789012",
        "userName": "IotEventsLambda-RoleForIotEvents-ABCD123DEF456"
      }
    }
  },
  "eventTime": "2019-02-07T23:53:23Z",
  "eventSource": "iotevents.amazonaws.com",
  "eventName": "ListDetectorModels",
  "awsRegion": "us-east-1",
  "sourceIPAddress": "192.168.0.1",
  "userAgent": "aws-internal/3",
  "requestParameters": {
    "nextToken": "CkZEZXRlY3Rvck1vZGVsMl9saXN0ZGV0ZWN0b3Jtb2RlbHN0ZXN0X2VlOWJkZTk1YT",
    "maxResults": 3
  },
  "responseElements": null,
  "requestID": "6d70f262-da95-4bb5-94b4-c08369df75bb",
  "eventID": "2d01a25c-d5c7-4233-99fe-ce1b8ec05516",
  "eventType": "AwsApiCall",
  "recipientAccountId": "123456789012"
}
```

例: CloudTrail の ListDetectorModelVersions アクション

以下の例は、`ListDetectorModelVersions` アクションを示す CloudTrail ログエントリです。

```
{
  "eventVersion": "1.05",
  "userIdentity": {
    "type": "AssumedRole",
    "principalId": "AKIAI44QH8DHBEXAMPLE:IotEvents-EventsLambda",
    "arn": "arn:aws:sts::123456789012:assumed-role/IotEventsLambda-RoleForIotEvents-
ABCD123DEF456/IotEvents-EventsLambda",
    "accountId": "123456789012",
    "accessKeyId": "AKIAI44QH8DHBEXAMPLE",
    "sessionContext": {
      "attributes": {
        "mfaAuthenticated": "false",
        "creationDate": "2019-02-07T22:22:30Z"
      },
```

```
      "sessionIssuer": {
        "type": "Role",
        "principalId": "AKIAI44QH8DHBEXAMPLE",
        "arn": "arn:aws:iam::123456789012:role/IotEventsLambda-RoleForIotEvents-
ABCD123DEF456",
        "accountId": "123456789012",
        "userName": "IotEventsLambda-RoleForIotEvents-ABCD123DEF456"
      }
    }
  },
  "eventTime": "2019-02-07T23:53:33Z",
  "eventSource": "iotevents.amazonaws.com",
  "eventName": "ListDetectorModelVersions",
  "awsRegion": "us-east-1",
  "sourceIPAddress": "192.168.0.1",
  "userAgent": "aws-internal/3",
  "requestParameters": {
    "detectorModelName": "myDetectorModel",
    "maxResults": 2
  },
  "responseElements": null,
  "requestID": "ebecb277-6bd8-44ea-8abd-fbf40ac044ee",
  "eventID": "fc6281a2-3fac-4e1e-98e0-ca6560b8b8be",
  "eventType": "AwsApiCall",
  "recipientAccountId": "123456789012"
}
```

例: CloudTrail の ListDetectors アクション

以下の例は、`ListDetectors` アクションを示す CloudTrail ログエントリです。

```
{
  "eventVersion": "1.05",
  "userIdentity": {
    "type": "AssumedRole",
    "principalId": "AKIAI44QH8DHBEXAMPLE:IotEvents-EventsLambda",
    "arn": "arn:aws:sts::123456789012:assumed-role/IotEventsLambda-RoleForIotEvents-
ABCD123DEF456/IotEvents-EventsLambda",
    "accountId": "123456789012",
    "accessKeyId": "AKIAI44QH8DHBEXAMPLE",
    "sessionContext": {
      "attributes": {
        "mfaAuthenticated": "false",
        "creationDate": "2019-02-07T22:22:30Z"
```

```
      },
      "sessionIssuer": {
        "type": "Role",
        "principalId": "AKIAI44QH8DHBEXAMPLE",
        "arn": "arn:aws:iam::123456789012:role/IotEventsLambda-RoleForIotEvents-
ABCD123DEF456",
        "accountId": "123456789012",
        "userName": "IotEventsLambda-RoleForIotEvents-ABCD123DEF456"
      }
    }
  },
  "eventTime": "2019-02-07T23:53:54Z",
  "eventSource": "iotevents.amazonaws.com",
  "eventName": "ListDetectors",
  "awsRegion": "us-east-1",
  "sourceIPAddress": "192.168.0.1",
  "userAgent": "aws-internal/3",
  "requestParameters": {
    "detectorModelName": "batchputmessagedetectorinstancecreated",
    "stateName": "HIDDEN_DUE_TO_SECURITY_REASONS"
  },
  "responseElements": null,
  "requestID": "4783666d-1e87-42a8-85f7-22d43068af94",
  "eventID": "0d2b7e9b-afe6-4aef-afd2-a0bb1e9614a9",
  "eventType": "AwsApiCall",
  "recipientAccountId": "123456789012"
}
```

例: CloudTrail の ListInputs アクション

以下の例は、`ListInputs` アクションを示す CloudTrail ログエントリです。

```
{
  "eventVersion": "1.05",
  "userIdentity": {
    "type": "AssumedRole",
    "principalId": "AKIAI44QH8DHBEXAMPLE:IotEvents-EventsLambda",
    "arn": "arn:aws:sts::123456789012:assumed-role/IotEventsLambda-RoleForIotEvents-
ABCD123DEF456/IotEvents-EventsLambda",
    "accountId": "123456789012",
    "accessKeyId": "AKIAI44QH8DHBEXAMPLE",
    "sessionContext": {
      "attributes": {
        "mfaAuthenticated": "false",
```

```
        "creationDate": "2019-02-07T22:22:30Z"
      },
      "sessionIssuer": {
        "type": "Role",
        "principalId": "AKIAI44QH8DHBEXAMPLE",
        "arn": "arn:aws:iam::123456789012:role/IotEventsLambda-RoleForIotEvents-
ABCD123DEF456",
        "accountId": "123456789012",
        "userName": "IotEventsLambda-RoleForIotEvents-ABCD123DEF456"
      }
    }
  },
  "eventTime": "2019-02-07T23:53:57Z",
  "eventSource": "iotevents.amazonaws.com",
  "eventName": "ListInputs",
  "awsRegion": "us-east-1",
  "sourceIPAddress": "192.168.0.1",
  "userAgent": "aws-internal/3",
  "requestParameters": {
    "nextToken": "CkhjYW5hcnlfdGVzdF9pbnB1dF9saXN0ZGV0ZWN0b3Jtb2RlbHN0ZXN0ZDU3OGZ",
    "maxResults": 3
  },
  "responseElements": null,
  "requestID": "dd6762a1-1f24-4e63-a986-5ea3938a03da",
  "eventID": "c500f6d8-e271-4366-8f20-da4413752469",
  "eventType": "AwsApiCall",
  "recipientAccountId": "123456789012"
}
```

例: CloudTrail の PutLoggingOptions アクション

以下の例は、PutLoggingOptions アクションを示す CloudTrail ログエントリです。

```
{
  "eventVersion": "1.05",
  "userIdentity": {
    "type": "AssumedRole",
    "principalId": "AKIAI44QH8DHBEXAMPLE:IotEvents-EventsLambda",
    "arn": "arn:aws:sts::123456789012:assumed-role/IotEventsLambda-RoleForIotEvents-
ABCD123DEF456/IotEvents-EventsLambda",
    "accountId": "123456789012",
    "accessKeyId": "AKIAI44QH8DHBEXAMPLE",
    "sessionContext": {
      "attributes": {
```

```
        "mfaAuthenticated": "false",
        "creationDate": "2019-02-07T22:22:30Z"
      },
      "sessionIssuer": {
        "type": "Role",
        "principalId": "AKIAI44QH8DHBEXAMPLE",
        "arn": "arn:aws:iam::123456789012:role/IotEventsLambda-RoleForIotEvents-
ABCD123DEF456",
        "accountId": "123456789012",
        "userName": "IotEventsLambda-RoleForIotEvents-ABCD123DEF456"
      }
    }
  },
  "eventTime": "2019-02-07T23:56:43Z",
  "eventSource": "iotevents.amazonaws.com",
  "eventName": "PutLoggingOptions",
  "awsRegion": "us-east-1",
  "sourceIPAddress": "192.168.0.1",
  "userAgent": "aws-internal/3",
  "requestParameters": {
    "loggingOptions": {
      "roleArn": "arn:aws:iam::123456789012:role/logging__logging_role",
      "level": "INFO",
      "enabled": false
    }
  },
  "responseElements": null,
  "requestID": "df570e50-fb19-4636-9ec0-e150a94bc52c",
  "eventID": "3247f928-26aa-471e-b669-e4a9e6fbc42c",
  "eventType": "AwsApiCall",
  "recipientAccountId": "123456789012"
}
```

例: CloudTrail の UpdateDetectorModel アクション

以下の例は、`UpdateDetectorModel` アクションを示す CloudTrail ログエントリです。

```
{
  "eventVersion": "1.05",
  "userIdentity": {
    "type": "AssumedRole",
    "principalId": "AKIAI44QH8DHBEXAMPLE:IotEvents-EventsLambda",
    "arn": "arn:aws:sts::123456789012:assumed-role/IotEventsLambda-RoleForIotEvents-
ABCD123DEF456/IotEvents-EventsLambda",
```

```
    "accountId": "123456789012",
    "accessKeyId": "AKIAI44QH8DHBEXAMPLE",
    "sessionContext": {
      "attributes": {
        "mfaAuthenticated": "false",
        "creationDate": "2019-02-07T22:22:30Z"
      },
      "sessionIssuer": {
        "type": "Role",
        "principalId": "AKIAI44QH8DHBEXAMPLE",
        "arn": "arn:aws:iam::123456789012:role/IotEventsLambda-RoleForIotEvents-
ABCD123DEF456",
        "accountId": "123456789012",
        "userName": "IotEventsLambda-RoleForIotEvents-ABCD123DEF456"
      }
    }
  },
  "eventTime": "2019-02-07T23:55:51Z",
  "eventSource": "iotevents.amazonaws.com",
  "eventName": "UpdateDetectorModel",
  "awsRegion": "us-east-1",
  "sourceIPAddress": "192.168.0.1",
  "userAgent": "aws-internal/3",
  "requestParameters": {
    "detectorModelName": "myDetectorModel",
    "roleArn": "arn:aws:iam::123456789012:role/Events_action_execution_role"
  },
  "responseElements": null,
  "requestID": "add29860-c1c5-4091-9917-d2ef13c356cf",
  "eventID": "7baa9a14-6a52-47dc-aea0-3cace05147c3",
  "eventType": "AwsApiCall",
  "recipientAccountId": "123456789012"
}
```

例: CloudTrail の UpdateInput アクション

以下の例は、`UpdateInput` アクションを示す CloudTrail ログエントリです。

```
{
  "eventVersion": "1.05",
  "userIdentity": {
    "type": "AssumedRole",
    "principalId": "AKIAI44QH8DHBEXAMPLE:IotEvents-EventsLambda",
```

```
    "arn": "arn:aws:sts::123456789012:assumed-role/IotEventsLambda-RoleForIotEvents-
ABCD123DEF456/IotEvents-EventsLambda",
    "accountId": "123456789012",
    "accessKeyId": "AKIAI44QH8DHBEXAMPLE",
    "sessionContext": {
      "attributes": {
        "mfaAuthenticated": "false",
        "creationDate": "2019-02-07T22:22:30Z"
      },
      "sessionIssuer": {
        "type": "Role",
        "principalId": "AKIAI44QH8DHBEXAMPLE",
        "arn": "arn:aws:iam::123456789012:role/IotEventsLambda-RoleForIotEvents-
ABCD123DEF456",
        "accountId": "123456789012",
        "userName": "IotEventsLambda-RoleForIotEvents-ABCD123DEF456"
      }
    }
  },
  "eventTime": "2019-02-07T23:53:00Z",
  "eventSource": "iotevents.amazonaws.com",
  "eventName": "UpdateInput",
  "awsRegion": "us-east-1",
  "sourceIPAddress": "192.168.0.1",
  "userAgent": "aws-internal/3",
  "errorCode": "ResourceNotFoundException",
  "errorMessage": "Input of name: NoSuchInput not found",
  "requestParameters": {
    "inputName": "NoSuchInput",
    "inputDescription": "this is a description of an input"
  },
  "responseElements": null,
  "requestID": "58d5d2bb-4110-4c56-896a-ee9156009f41",
  "eventID": "c2df241a-fd53-4fd0-936c-ba309e5dc62d",
  "eventType": "AwsApiCall",
  "recipientAccountId": "123456789012"
}
```

例: CloudTrail の BatchPutMessage アクション

AWS IoT Events は、データプレーン API ログ記録に CloudTrail 統合を使用できます。この例では、`BatchPutMessage`アクションを通じてデータイベントの詳細を追加します。

```
{
            "eventVersion": "1.09",
            "userIdentity": {
                "type": "AssumedRole",
                "principalId": "AKIAI44QH8DHBEXAMPLE:PrincipalId",
                "arn": "arn:aws:sts::123456789012:assumed-role/my-iam-role/my-iam-role-
entity",
                "accountId": "123456789012",
                "accessKeyId": "AKIAI44QH8DHBEXAMPLE",
                "sessionContext": {
                    "sessionIssuer": {
                        "type": "Role",
                        "principalId": "AKIAI44QH8DHBEXAMPLE",
                        "arn": "arn:aws:iam::123456789012:role/my-iam-role",
                        "accountId": "123456789012",
                        "userName": "sample_user_name"
                    },
                    "attributes": {
                        "creationDate": "2024-11-22T18:32:41Z",
                        "mfaAuthenticated": "false"
                    }
                }
            },
            "eventTime": "2024-11-22T18:57:35Z",
            "eventSource": "iotevents.amazonaws.com",
            "eventName": "BatchPutMessage",
            "awsRegion": "us-east-1",
            "sourceIPAddress": "3.239.107.128",
            "userAgent": "aws-internal/3",
            "requestParameters": {
                "messages": [
                    {
                        "messageId": "e306d827-b2e4-4439-9c86-411d4242a397",
                        "payload": "HIDDEN_DUE_TO_SECURITY_REASONS",
                        "inputName": "my_input_name"
                    }
                ]
            },
            "responseElements": {
                "batchPutMessageErrorEntries": []
            },
            "requestID": "cefc6b63-9ccf-4e31-9177-4aec8e701bfe",
            "eventID": "b994b52c-6011-4e3c-ad5f-e784e732fde0",
```

```
            "readOnly": false,
            "resources": [
                {
                    "accountId": "123456789012",
                    "type": "AWS::IoTEvents::Input",
                    "ARN": "arn:aws:iotevents:us-east-1:123456789012:input/
my_input_name"
                }
            ],
            "eventType": "AwsApiCall",
            "managementEvent": false,
            "recipientAccountId": "123456789012",
            "eventCategory": "Data",
            "tlsDetails": {
                "tlsVersion": "TLSv1.3",
                "cipherSuite": "TLS_AES_128_GCM_SHA256",
                "clientProvidedHostHeader": "iotevents.us-east-1.amazonaws.com"
            }
        },
```

のコンプライアンス検証 AWS IoT Events

AWS のサービス が特定のコンプライアンスプログラムの範囲内にあるかどうかを確認するには、コンプライアンスAWS のサービス プログラムによる範囲内コンプライアンスを参照し、関心のあるコンプライアンスプログラムを選択します。一般的な情報については、AWS 「 Compliance ProgramsAssurance」を参照してください。

を使用して、サードパーティーの監査レポートをダウンロードできます AWS Artifact。詳細については、「Downloading AWS Artifact Reports 」を参照してください。

を使用する際のお客様のコンプライアンス責任 AWS のサービス は、お客様のデータの機密性、貴社のコンプライアンス目的、適用される法律および規制によって決まります。 では、コンプライアンスに役立つ以下のリソース AWS を提供しています。

- セキュリティのコンプライアンスとガバナンス – これらのソリューション実装ガイドでは、アーキテクチャ上の考慮事項について説明し、セキュリティとコンプライアンスの機能をデプロイする手順を示します。
- HIPAA 対応サービスのリファレンス – HIPAA 対応サービスの一覧が提供されています。すべてが HIPAA 対応 AWS のサービス であるわけではありません。

- AWS コンプライアンスリソース – このワークブックとガイドのコレクションは、お客様の業界や地域に適用される場合があります。
- AWS カスタマーコンプライアンスガイド – コンプライアンスの観点から責任共有モデルを理解します。このガイドは、複数のフレームワーク (米国国立標準技術研究所 (NIST)、Payment Card Industry Security Standards Council (PCI)、国際標準化機構 (ISO) など) にわたるセキュリティコントロールを保護し、そのガイダンスに AWS のサービス マッピングするためのベストプラクティスをまとめたものです。
- 「 デベロッパーガイド」の「ルールによるリソースの評価」 – この AWS Config サービスは、リソース設定が社内プラクティス、業界ガイドライン、および規制にどの程度準拠しているかを評価します。 AWS Config
- AWS Security Hub – これにより AWS のサービス 、セキュリティ状態を包括的に把握できます AWS。Security Hub では、セキュリティコントロールを使用して AWS リソースを評価し、セキュリティ業界標準とベストプラクティスに対するコンプライアンスをチェックします。サポートされているサービスとコントロールの一覧については、Security Hub のコントロールリファレンスを参照してください。
- Amazon GuardDuty – 環境をモニタリングして AWS アカウント不審なアクティビティや悪意のあるアクティビティがないか調べることで、、ワークロード、コンテナ、データに対する潜在的な脅威 AWS のサービス を検出します。GuardDuty を使用すると、特定のコンプライアンスフレームワークで義務付けられている侵入検知要件を満たすことで、PCI DSS などのさまざまなコンプライアンス要件に対応できます。
- AWS Audit Manager – これにより AWS のサービス 、 AWS 使用状況を継続的に監査し、リスクの管理方法と規制や業界標準への準拠を簡素化できます。

の耐障害性 AWS IoT Events

AWS グローバルインフラストラクチャは、 AWS リージョンとアベイラビリティーゾーンを中心に構築されています。 AWS リージョンには、低レイテンシー、高いスループット、そして高度の冗長ネットワークで接続されている複数の物理的に独立し隔離されたアベイラビリティーゾーンがあります。アベイラビリティーゾーンでは、アベイラビリティーゾーン間で中断せずに、自動的にフェイルオーバーするアプリケーションとデータベースを設計および運用することができます。アベイラビリティーゾーンは、従来の単一または複数のデータセンターインフラストラクチャよりも可用性、耐障害性、およびスケーラビリティが優れています。

AWS リージョンとアベイラビリティーゾーンの詳細については、AWS 「 グローバルインフラストラクチャ」を参照してください。

のインフラストラクチャセキュリティ AWS IoT Events

マネージドサービスである AWS IoT Events は、 AWS グローバルネットワークセキュリティで保護されています。 AWS セキュリティサービスと がインフラストラクチャ AWS を保護する方法については、AWS 「 クラウドセキュリティ」を参照してください。インフラストラクチャセキュリティのベストプラクティスを使用して AWS 環境を設計するには、「Security Pillar AWS Well-Architected Framework」の「Infrastructure Protection」を参照してください。

AWS が公開した API コールを使用して、ネットワーク AWS IoT Events 経由で にアクセスします。クライアントは以下をサポートする必要があります。

- Transport Layer Security (TLS)。TLS 1.2 が必須で、TLS 1.3 をお勧めします。
- DHE (楕円ディフィー・ ヘルマン鍵共有) や ECDHE (楕円曲線ディフィー・ ヘルマン鍵共有) などの完全前方秘匿性 (PFS) による暗号スイート。これらのモードはJava 7 以降など、ほとんどの最新システムでサポートされています。

また、リクエストにはアクセスキー ID と、IAM プリンシパルに関連付けられているシークレットアクセスキーを使用して署名する必要があります。または、AWS Security Token ServiceAWS STSを使用して、一時的なセキュリティ認証情報を生成し、リクエストに署名することもできます。

AWSAWS IoT Events リソースのサービスクォータ

AWS 全般のリファレンス ガイドには、 AWS アカウントの AWS IoT Events のデフォルトのクォータが記載されています。指定されていない限り、各クォータは AWS リージョンごとです。詳細については、「AWS 全般のリファレンス ガイド」の「AWS IoT Events エンドポイントとクォータ」および「AWS Service Quotas」を参照してください。

サービスクォータ の増加を要求するには、サポートセンターコンソールでサポートケースを送信します。詳細については、「Service Quotas ユーザーガイド」の「クォータ引き上げのリクエスト」を参照してください。

> **Note**
>
> - ディテクターモデルと入力の名前はすべてアカウント内で一意である必要があります。
> - ディテクターモデルおよび入力の名前は、作成後変更することはできません。

AWS IoT Events リソースのタグ付け

デテクターモデルと入力の管理と整理を支援するために、オプションで、タグの形式でこれらの各リソースに独自のメタデータを割り当てることができます。このセクションでは、タグとその作成方法について説明します。

ベーシックタグ

タグを使用すると、 AWS IoT Events リソースを目的、所有者、環境などさまざまな方法で分類できます。これは、同じ種類のリソースが多い場合に役立ちます。リソースに割り当てたタグに基づいて、特定のリソースをすばやく識別できます。

タグはそれぞれ、1 つのキーとオプションの値で設定され、どちらもユーザーが定義します。例えば、入力のタグのセットを定義して、これらの入力を送信するデバイスをタイプ別に追跡するのに役立てることができます。リソースの種類ごとのニーズを合わせて一連のタグキーを作成することをお勧めします。一貫性のあるタグキーセットを使用することで、リソースの管理が容易になります。

AWS IoT デベロッパーガイドのIAM ポリシーでのタグの使用で説明されているように、追加または適用するタグに基づいてリソースを検索およびフィルターリングしたり、タグを使用してコストを分類および追跡したり、タグを使用してリソースへのアクセスを制御したりできます。

使いやすさのために、 のタグエディタ AWS Management Console は、タグを作成および管理するための一元的な統一された方法を提供します。詳細については、「 リソースのタグ付け」および「タグエディタユーザーガイド」の「タグエディタの開始方法」を参照してください。 AWS

AWS CLI および AWS IoT Events API を使用してタグを操作することもできます。次のコマンドで "Tags" フィールドを使用して、タグを作成するときに、タグをディテクターモデルおよび入力に関連付けることができます。

- CreateDetectorModel
- CreateInput

以下のコマンドを使用して、タグ付けがサポートされている既存のリソースに対してタグを追加、変更、または削除できます。

- TagResource
- ListTagsForResource

- UntagResource

タグのキーと値は編集でき、タグはリソースからいつでも削除できます。タグの値を空の文字列に設定することはできますが、タグの値を null に設定することはできません。特定のリソースについて既存のタグと同じキーを持つタグを追加した場合、古い値は新しい値によってオーバーライドされます。リソースを削除すると、リソースに関連付けられているすべてのタグも削除されます。

詳細については、「AWS リソースのタグ付けのベストプラクティス」を参照してください。

タグの制約と制限

次のベーシックな制限がタグに適用されます。

- リソースあたりのタグの最大数 - 50
- キーの最大長 - UTF-8 の 127 Unicode 文字
- 値の最大長 - UTF-8 の 255 Unicode 文字
- タグのキーと値では、大文字と小文字が区別されます。
- タグ名または値に `"aws:"` プレフィックスを使用しないでください。このプレフィックスは AWS 用に予約されています。このプレフィックスが含まれるタグの名前または値は編集または削除できません。このプレフィックスを持つタグは、リソースあたりのタグ数の制限にはカウントされません。
- 複数のサービス間およびリソース間でタグ付けスキーマを使用する場合、他のサービスでも許可される文字に制限が適用されることがあるのでご注意ください。一般に、許可される文字は、UTF-8 で表現可能な文字、スペース、数字、および次の特殊文字です: + - = . _ : / @。

IAM ポリシーでのタグの使用

AWS IoT Events API アクションに対して使用する IAM ポリシーで、タグベースのリソースレベルアクセス許可を適用することができます。これにより、ユーザーがどのリソースを作成、変更、または使用できるかを制御しやすくなります。

IAM ポリシーの以下の条件コンテキストのキーと値とともに `Condition` 要素 (`Condition` ブロックとも呼ばれる) を使用して、リソースのタグに基づいてユーザーアクセス (アクセス許可) を制御できます。

- 特定のタグを持つリソースに対してユーザーアクションを許可または拒否するには、`aws:ResourceTag/<tag-key>: <tag-value>` を使用します。

- タグが許可されているリソースを作成または変更する API リクエストを作成する場合に、特定のタグが使用されている (または、使用されていない) ことを要求するには、`aws:RequestTag/<tag-key>: <tag-value>` を使用します。
- タグが許可されているリソースを作成または変更する API リクエストを作成する場合に、特定の一連のタグが使用されている (または、使用されていない) ことを要求するには、`aws:TagKeys: [<tag-key>, ...]` を使用します。

> **Note**
>
> IAM ポリシーの条件コンテキストのキーと値は、タグ付け可能なリソースの ID が必須パラメータである AWS IoT Events アクションにのみ適用されます。

AWS Identity and Access Management ユーザーガイドのタグを使用したアクセスの制御には、タグの使用に関する追加情報があります。そのガイドの [IAM JSON policy reference] (IAM JSON ポリシーリファレンス) セクションには、IAM での JSON ポリシーの要素、可変、および評価ロジックの詳細な構文、説明、および例が記載されています。

次のポリシー例では、タグベースの 2 つの制約が適用されています。このポリシーによって制限されているユーザー

- リソースにタグ「env=pro」を付けることはできません (例では、 `"aws:RequestTag/env" : "prod"` 行を参照してください)
- 既存のタグ「env=prod」を持つリソースを変更またはアクセスできません (例では、`"aws:ResourceTag/env" : "prod"` 行を参照してください)。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Deny",
            "Action": [
                "iotevents:CreateDetectorModel",
                "iotevents:CreateAlarmModel",
                "iotevents:CreateInput",
                "iotevents:TagResource"
            ],
            "Resource": "*",
```

```
            "Condition": {
                "StringEquals": {
                    "aws:RequestTag/env": "prod"
                }
            }
        },
        {
            "Effect": "Deny",
            "Action": [
                "iotevents:DescribeDetectorModel",
                "iotevents:DescribeAlarmModel",
                "iotevents:UpdateDetectorModel",
                "iotevents:UpdateAlarmModel",
                "iotevents:DeleteDetectorModel",
                "iotevents:DeleteAlarmModel",
                "iotevents:ListDetectorModelVersions",
                "iotevents:ListAlarmModelVersions",
                "iotevents:UpdateInput",
                "iotevents:DescribeInput",
                "iotevents:DeleteInput",
                "iotevents:ListTagsForResource",
                "iotevents:TagResource",
                "iotevents:UntagResource",
                "iotevents:UpdateInputRouting"
            ],
            "Resource": "*",
            "Condition": {
                "StringLike": {
                    "aws:ResourceTag/env": "prod"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "iotevents:*"
            ],
            "Resource": "*"
        }
    ]
}
```

次のように、リストで囲むことにより、特定のタグキーに複数のタグ値を指定することもできます。

```
            "StringEquals" : {
              "aws:ResourceTag/env" : ["dev", "test"]
            }
```

> **Note**
>
> タグに基づいてリソースへのユーザーのアクセスを許可または拒否する場合は、ユーザーが同じリソースに対してそれらのタグを追加または削除することを明示的に拒否することを検討する必要があります。そうしないと、ユーザーはそのリソースのタグを変更することで、制限を回避してリソースにアクセスできてしまいます。

トラブルシューティング AWS IoT Events

このトラブルシューティングガイドでは、 の使用時に発生する可能性がある一般的な問題の解決策を示します AWS IoT Events。トピックを参照して、イベントの検出、データへのアクセス、アクセス許可、サービス統合、デバイス設定などの問題を特定して解決します。 AWS IoT Events コンソール、API、CLI、エラー、レイテンシー、統合に関するトラブルシューティングのアドバイスにより、このガイドは、信頼性が高くスケーラブルなイベント駆動型アプリケーションを構築できるように、問題を迅速に解決することを目指しています。

トピック

- 一般的な AWS IoT Events 問題と解決策
- で分析を実行してディテクターモデルのトラブルシューティングを行う AWS IoT Events

一般的な AWS IoT Events 問題と解決策

次のセクションを参照して、エラーのトラブルシューティングを行い、問題を解決するための考えられる解決策を見つけます AWS IoT Events。

エラー

- ディテクターモデルの作成エラー
- 削除されたディテクターモデルからの更新
- アクショントリガーの障害 (条件を満たす場合)
- アクショントリガーの障害 (しきい値を超えた場合)
- 誤った状態の使用法
- 接続メッセージ
- InvalidRequestException メッセージ
- Amazon CloudWatch Logs action.setTimer エラー
- Amazon CloudWatch ペイロードエラー
- 互換性のないデータ型
- へのメッセージの送信に失敗しました AWS IoT Events

ディテクターモデルの作成エラー

ディテクターモデルを作成しようとする時にエラーが発生します。

ソリューション

ディテクターモデルを作成する時は、次の制限を考慮する必要があります。

- 各 `action` フィールドで許可されるアクションは 1 つだけです。
- `transitionEvents` には `condition` が必要です。`OnEnter`、`OnInput`、および `OnExit` イベントではオプションです。
- `condition` フィールドが空の場合、条件表現の評価結果は `true` と同等です。
- 条件表現の評価結果はブール値である必要があります。結果がブール値でない場合、`false` と同等であり、`actions` をトリガーしたり、イベントで指定された `nextState` に移行したりすることはありません。

詳細については、「AWS IoT Events ディテクターモデルの制約と制限」を参照してください。

削除されたディテクターモデルからの更新

数分前にディテクターモデルを更新または削除しましたが、MQTT メッセージまたは SNS アラートを介して古いディテクターモデルから状態の更新を取得しています。

ソリューション

ディテクターモデルを更新、削除、または再作成する場合 (UpdateDetectorModelを参照)、すべてのディテクターインスタンスが削除されて新しいモデルが使用されるまでに遅延が発生します。この間、入力は以前のバージョンのディテクターモデルのインスタンスによって処理され続ける可能性があります。以前のディテクターモデルで定義されたアラートを引き続き受信する場合があります。更新を再確認するか、エラーを報告する前に、少なくとも 7 分間待ちます。

アクショントリガーの障害 (条件を満たす場合)

条件が満たされた場合、ディテクターはアクションのトリガーまたは新しい状態への移行に失敗します。

ソリューション

ディテクターの条件表現の評価結果がブール値であることを確認します。結果がブール値でない場合、false と同等であり、action をトリガーしたり、イベントで指定された nextState に移行したりすることはありません。詳細については、「条件表現の構文」を参照してください。

アクショントリガーの障害 (しきい値を超えた場合)

条件表現の可変が指定された値に達した場合、ディテクターはアクションまたはイベント移行をトリガーしません。

ソリューション

onInput、onEnter、または onExit の setVariable を更新する場合、現在の処理サイクル中に condition を評価するときに、新しい値は使用されません。代わりに、現在のサイクルが完了するまで元の値が使用されます。この動作は、ディテクターモデル定義で evaluationMethod パラメータを設定することで変更できます。evaluationMethod が SERIAL に設定されている場合、可変が更新され、イベントが定義された順序でイベント条件が評価されます。evaluationMethod が BATCH (デフォルト) に設定されている場合、可変が更新され、すべてのイベント条件が評価された後にのみイベントが実行されます。

誤った状態の使用法

BatchPutMessage を使用して入力にメッセージを送信しようとすると、ディテクターが間違った状態になります。

ソリューション

BatchPutMessage を使用して複数のメッセージを入力に送信する場合、メッセージまたは入力が処理される順序は保証されません。順序付けを保証するには、一度に 1 つずつメッセージを送信し、BatchPutMessage が成功を確認するたびに待ちます。

接続メッセージ

API を呼び出したり呼び出したりしようとすると、('Connection aborted.', error(54, 'Connection reset by peer')) エラーが発生します。

ソリューション

OpenSSL が TLS 1.1 以降のバージョンを使用して接続を確立していることを確認します。これは、ほとんどの Linux ディストリビューションまたは Windows バージョン 7 以降ではデフォルトになっ

ているはずです。macOS のユーザーは、OpenSSL をアップグレードする必要があるかもしれません。

InvalidRequestException メッセージ

`CreateDetectorModel` および `UpdateDetectorModel` API を呼び出そうとすると、InvalidRequestException が発生します。

ソリューション

問題の解決に役立つように、以下をチェックしてください。詳細については、「CreateDetectorModel」および「UpdateDetectorModel」を参照してください。

- `SetTimerAction` のパラメータとして `seconds` と `durationExpression` の両方を同時に使用しないように注意してください。
- `durationExpression` の文字列表現が有効であることを確認してください。文字列表現には、数値、可変 (`$variable.<variable-name>`)、または入力値 (`$input.<input-name>.<path-to-datum>`) を含めることができます。

Amazon CloudWatch Logs **`action.setTimer`** エラー

AWS IoT Events ディテクターモデルインスタンスをモニタリングするように Amazon CloudWatch Logs を設定できます。以下は、 を使用する場合 AWS IoT Eventsに によって生成される一般的なエラーです`action.setTimer`。

- エラー: *`<timer-name>`* という名前のタイマーの期間表現を数値に評価できませんでした。

ソリューション

`durationExpression` の文字列表現を数値に変換できることを確認してください。ブール値などの他のデータ型は許可されていません。

- エラー: *`<timer-name>`*という名前のタイマーの期間表現の評価結果が 31,622,440 を超えています。精度を確保するために、期間表現が 60〜31,622,400 の値を参照していることを確認してください。

ソリューション

タイマーの持続時間が 31,622,400 秒以下であることを確認してください。期間の評価結果は、最も近い整数に切り捨てられます。

- エラー: *<timer-name>* という名前のタイマーの期間表現の評価結果は 60 未満です。精度を確保するために、期間表現が 60〜31,622,400 の値を参照していることを確認してください。

ソリューション

タイマーの持続時間が 60 秒以上であることを確認してください。期間の評価結果は、最も近い整数に切り捨てられます。

- エラー: *<timer-name>*という名前のタイマーの期間表現を評価できませんでした。可変名、入力名、およびデータへのパスをチェックして、既存の可変と入力を参照していることを確認してください。

ソリューション

文字列表現が既存の可変と入力を参照していることを確認してください。文字列表現には、数値、可変 ($variable.*variable-name*)、および入力値 ($input.*input-name*.*path-to-datum*) を含めることができます。

- エラー: <timer-name>という名前のタイマーを設定できませんでした。持続時間の表現をチェックして、再試行してください。

ソリューション

SetTimerActionアクションを参照して、正しいパラメータを指定したことを確認してから、タイマーを再設定してください。

詳細については、「ディAWS IoT Events テクターモデルの開発時に Amazon CloudWatch ログ記録を有効にする」を参照してください。

Amazon CloudWatch ペイロードエラー

AWS IoT Events ディテクターモデルインスタンスをモニタリングするように Amazon CloudWatch Logs を設定できます。以下は、アクションペイロードを設定するときに AWS IoT Eventsによって生成される一般的なエラーと警告です。

- エラー: アクションの表現を評価できませんでした。可変名、入力名、およびデータへのパスが既存の可変と入力値を参照していることを確認してください。また、ペイロードのサイズが、ペイロードの最大許容サイズである 1 KB 未満であることを確認してください。

ソリューション

正しい可変名、入力名、およびデータへのパスを入力していることを確認してください。アクションペイロードが 1 KB より大きい場合にも、このエラーメッセージが表示されることがあります。

- エラー: `<action-type>` のペイロードのコンテンツ表現を分析できませんでした。正しい構文でコンテンツ表現を入力してください。

ソリューション

コンテンツ表現には、文字列 (`'string'`)、可変 (`$variable.variable-name`)、入力値 (`$input.input-name.path-to-datum`)、文字列の連結、および `${}` を含む文字列を含めることができます。

- エラー: ペイロード表現 {`##`} が無効です。定義されたペイロードタイプは JSON であるため、文字列に対して評価 AWS IoT Events される式を指定する必要があります。

ソリューション

指定されたペイロードタイプが JSON の場合、 AWS IoT Events は最初に、サービスが式を文字列に評価できるかどうかをチェックします。評価結果をブール値または数値にすることはできません。検証が失敗した場合、このエラーが発生する可能性があります。

- 警告: アクションは実行されましたが、有効な JSON に対するアクションペイロードのコンテンツ表現を評価できませんでした。定義されたペイロードタイプは JSON です。

ソリューション

ペイロードタイプを として定義した場合、 AWS IoT Events がアクションペイロードのコンテンツ式を有効な JSON に評価できることを確認しますJSON。 AWS IoT Events がコンテンツ式を有効な JSON に評価できない場合でも、 AWS IoT Events はアクションを実行します。

詳細については、「ディ[AWS IoT Events テクターモデルの開発時に Amazon CloudWatch ログ記録を有効にする](#)」を参照してください。

互換性のないデータ型

メッセージ: 互換性のないデータ型 [`<inferred-types>`] が次の表現の `<reference>` で見つかりました: `<expression>`

ソリューション

次のいずれかの理由でこのエラーが発生する可能性があります。

- リファレンスの評価結果は、表現の他のオペランドと互換性がありません。
- 関数に渡される引数の型はサポートされていません。

表現でリファレンスを使用する場合は、以下をチェックしてください。

- 1 つ以上の演算子でオペランドとしてリファレンスを使用する場合は、参照するすべてのデータ型に互換性があることを確認してください。

  例えば、次の表現では、整数 2 は == 演算子と && 演算子の両方のオペランドです。オペランドに互換性があることを確認するには、`$variable.testVariable + 1` と `$variable.testVariable` は整数または 10 進数をリファレンスする必要があります。

  また、整数 1 は + 演算子のオペランドです。したがって、`$variable.testVariable` は整数または 10 進数をリファレンスする必要があります。

  ```
  ‘$variable.testVariable + 1 == 2 && $variable.testVariable’
  ```

- 関数に渡される引数としてリファレンスを使用する場合は、関数が参照するデータ型をサポートしていることを確認してください。

  例えば、次の `timeout("time-name")` 関数では、引数として二重引用符を含む文字列が必要です。*timer-name* 値のリファレンスを使用する場合は、二重引用符で囲まれた文字列を参照する必要があります。

  ```
  timeout("timer-name")
  ```

  > **Note**
  >
  > `convert(type, expression)` 関数の場合、*###*値にリファレンスを使用する場合、リファレンスの評価結果は `String`、`Decimal`、または `Boolean` である必要があります。

詳細については、「AWS IoT Events 式の入力と変数の リファレンス」を参照してください。

へのメッセージの送信に失敗しました AWS IoT Events

メッセージ: IoT イベントにメッセージを送信できませんでした

ソリューション

このエラーは、次の理由で発生する場合があります。

- 入力メッセージのペイロードに `Input attribute Key` が含まれていない。
- `Input attribute Key` が、入力の定義で指定されているのと同じ JSON パス にない。
- 入力メッセージは、 AWS IoT Events 入力で定義されているスキーマと一致しません。

> **Note**
> 他のサービスからのデータインジェストにも障害が発生します。

Example

たとえば、 では AWS IoT Core、 AWS IoT ルールは次のメッセージで失敗します。 `Verify the Input Attribute key.`

これを解決するには、入力ペイロードメッセージスキーマが AWS IoT Events 入力定義に準拠し、`Input attribute Key`場所が一致していることを確認します。詳細については、でモデルの入力を作成する AWS IoT Events「」を参照して、 AWS IoT Events 入力を定義する方法を確認してください。

で分析を実行してディテクターモデルのトラブルシューティングを行う AWS IoT Events

AWS IoT Events は、入力データをディテクターモデルに送信せずに、ディテクターモデルを分析し、分析結果を生成できます。 は、このセクションで説明する一連の分析 AWS IoT Events を実行して、ディテクターモデルを確認します。このアドバンストなトラブルシューティングソリューションは、重要度レベルや場所などの診断情報も要約するため、ディテクターモデルの潜在的な問題をすばやく見つけて修正できます。ディテクターモデルの診断エラータイプとメッセージの詳細については、のディテクターモデル分析と診断情報 AWS IoT Events を参照してください。

AWS IoT Events コンソール、API、 AWS Command Line Interface（AWS CLI）、または AWS SDK を使用して、ディテクターモデルの分析からの診断エラーメッセージを表示できます。

> **Note**
>
> - ディテクターモデルを発行する前に、すべてのエラーを修正する必要があります。
> - 本番環境でディテクターモデルを使用する前に、警告を確認し、必要なアクションを実行することをお勧めします。そうしないと、ディテクターモデルが期待どおりに機能しない可能性があります。
> - 同時に RUNNING ステータスで最大 10 の分析を行うことができます。

ディテクターモデルの分析方法については、のディテクターモデルを分析する AWS IoT Events（コンソール）または AWS IoT Events（AWS CLI) でディテクターモデルを分析する を参照してください。

トピック

- のディテクターモデル分析と診断情報 AWS IoT Events
- のディテクターモデルを分析する AWS IoT Events（コンソール）
- AWS IoT Events（AWS CLI) でディテクターモデルを分析する

のディテクターモデル分析と診断情報 AWS IoT Events

ディテクターモデル分析は、次の診断情報を収集します。

- レベル - 分析結果の重要度レベル。重要度レベルに基づいて、分析結果は次の 3 つの一般的なカテゴリに分類されます。
  - 情報 (INFO) - 情報結果は、ディテクターモデルの重要なフィールドについての情報を提供します。このタイプの結果は通常、即時のアクションを必要としません。
  - 警告 (WARNING) - 警告結果は、ディテクターモデルに問題を引き起こす可能性のあるフィールドに注意を喚起します。本番環境でディテクターモデルを使用する前に、警告を再調査し、必要な行動をとることをお勧めします。そうしないと、ディテクターモデルが期待どおりに動作しない可能性があります。

- エラー (ERROR) - エラー結果は、ディテクターモデルで見つかった問題について通知します。ディテクターモデルを発行しようとする時、 AWS IoT Events はこの一連の分析を自動的に実行します。ディテクターモデルを発行する前に、すべてのエラーを修正する必要があります。

- 場所 - ディテクターモデル内の、分析結果が参照するフィールドを特定するための情報です。場所には、通常、状態名、移行イベント名、イベント名、そして表現 (例: `in state TemperatureCheck in onEnter in event Init in action setVariable`) が含まれます。

- タイプ- 分析結果のタイプ。分析タイプは、以下のように分類されます。

  - `supported-actions` – 指定されたイベントまたは移行イベントが検出されたときにアクションを呼び出す AWS IoT Events ことができます。組み込みアクションを定義して、タイマーを使用したり、可変を設定したり、他の AWS サービスにデータを送信したりできます。 AWS サービスが利用可能な AWS リージョンで、他の AWS サービスと連携するアクションを指定する必要があります。

  - `service-limits` – 制限とも呼ばれるサービスクォータは、 AWS アカウントのサービスリソースまたはオペレーションの最大数または最小数です。特に明記されていない限り、クォータはリージョンごとに決まっています。ビジネスニーズに応じて、ディテクターモデルを更新して、制限に遭遇しないようにするか、クォータの増加を要求できます。一部のクォータの増加を要求できますが、他のクォータは増加できません。詳細については、「クォータ」を参照してください。

- **`structure`** - ディテクターモデルは、状態などの必須な要素をすべて備え、 AWS IoT Events のサポートする構造に従わなければなりません。ディテクターモデルには、少なくとも 1 つの状態と 1 つの条件が必要です。条件は、重要なイベントを検出するために入力データを評価します。イベントが検出されると、ディテクターモデルは次の状態に移行し、アクションを呼び出すことができます。これらのイベントは、移行イベントと呼ばれます。移行イベントは、次の状態に入るように指示する必要があります。

- **`expression-syntax`** - AWS IoT Events では、ディテクターモデルの作成および更新時に値を指定する方法がいくつか用意されています。表現では、リテラル、演算子、関数、リファレンス、および置換テンプレートを使用できます。式を使用してリテラル値を指定するか、特定の値を指定する前に式を評価 AWS IoT Events できます。表現は、必要な構文に従う必要があります。詳細については、「イベントデータをフィルタリング、変換、処理する式」を参照してください。

  のディテクターモデル式は、特定のデータまたはリソースを参照 AWS IoT Events できます。

  - **`data-type`** - AWS IoT Events は、整数、10 進数、文字列、およびブールデータ型をサポートします。式の評価中に が 1 つのデータ型のデータを別のデータ型に自動的に変換 AWS IoT Events できる場合、これらのデータ型には互換性があります。

> **Note**
>
> - 整数と 10 進数は、 AWS IoT Eventsでサポートされている唯一の互換性のあるデータ型です。
> - AWS IoT Events は整数を文字列に変換できないため AWS IoT Events 、 は算術式を評価できません。

- **`referenced-data`** - データを使用する前に、ディテクターモデルで参照されるデータを定義しておく必要があります。例えば、DynamoDB テーブルにデータを送信する場合は、表現 (`$variable.TableName`) で可変を使用する前に、テーブル名を参照する可変を定義する必要があります。
- **`referenced-resource`** - ディテクターモデルが使用するリソースが使用可能である必要があります。リソースを使用する前に、リソースを定義する必要があります。例えば、温室の温度をモニタリングするためのディテクターモデルを作成するとします。`$input.TemperatureInput.sensorData.temperature` を使用して温度をリファレンスする前に、入力 (`$input.TemperatureInput`) を定義して、入力温度データをディテクターモデルに送信する必要があります。

次のセクションを参照して、エラーのトラブルシューティングを行い、ディテクターモデルの分析から考えられる解決策を見つけてください。

でのディテクターモデルエラーのトラブルシューティング AWS IoT Events

上記のタイプのエラーは、ディテクターモデルに関する診断情報を提供し、取得する可能性のあるメッセージに対応します。これらのメッセージと推奨される解決策を使用して、ディテクターモデルのエラーをトラブルシューティングします。

メッセージと解決策

- Location
- supported-actions
- service-limits
- structure
- expression-syntax
- data-type
- referenced-data

- referenced-resource

Location

`Location` に関する情報を含む分析結果は、次のエラーメッセージに対応します。

- メッセージ - 分析結果に関する追加情報が含まれています。これは、情報、警告、またはエラーメッセージである可能性があります。

  解決策: AWS IoT Events が現在サポートしていないアクションを指定した場合、このエラーメッセージが表示されることがあります。サポートされているアクションのリストについては、でデータを受信してアクションをトリガーするためにサポートされているアクション AWS IoT Events を参照してください。

supported-actions

`supported-actions` に関する情報を含む分析結果は、次のエラーメッセージに対応します。

- メッセージ: アクション定義*#######*に無効なアクションタイプが存在します。

  解決策: AWS IoT Events が現在サポートしていないアクションを指定した場合、このエラーメッセージが表示されることがあります。サポートされているアクションのリストについては、でデータを受信してアクションをトリガーするためにサポートされているアクション AWS IoT Events を参照してください。
- メッセージ: ディテクターモデル定義に *aws-service* アクションがありますが、リージョン*#######*で *aws-service* サービスがサポートされていません。

  解決策： 指定したアクションが でサポートされているが AWS IoT Events、アクションが現在のリージョンで使用できない場合に、このエラーメッセージが表示されることがあります。これは、リージョンで利用できない AWS サービスにデータを送信しようとしたときに発生する可能性があります。また、 AWS IoT Events と AWS 使用しているサービスの両方で同じリージョンを選択する必要があります。

service-limits

`service-limits` に関する情報を含む分析結果は、次のエラーメッセージに対応します。

- メッセージ: ペイロードで許可されているコンテンツ表現が、状態*###*のイベント*#####*の制限*########*バイトを超えました。

解決策: アクションペイロードのコンテンツ表現が 1,024 バイトを超える場合、このエラーメッセージが表示されることがあります。ペイロードのコンテンツ表現のサイズは、最大 1024 バイトです。

- メッセージ: ディテクターモデル定義で許可されている状態の数が、##############の制限を超えました。

  解決策: ディテクターモデルに 20 を超える状態がある場合、このエラーメッセージが表示されることがあります。ディテクターモデルには、最大 20 の状態を含めることができます。

- メッセージ: タイマー#####持続時間は、少なくとも########## 秒の長さである必要があります。

  解決策: タイマー持続時間が 60 秒未満の場合、このエラーメッセージが表示されることがあります。タイマー持続時間は 60〜31,622,400 秒にすることをお勧めします。タイマー持続時間の表現を指定すると、持続時間の表現の評価結果は最も近い整数に切り捨てられます。

- メッセージ: イベントごとに許可されるアクションの数が、ディテクターモデル定義の##########の制限を超えました

  解決策: イベントに 10 を超えるアクションがある場合、このエラーメッセージが表示されることがあります。ディテクターモデルでは、イベントごとに最大 10 のアクションを実行できます。

- メッセージ: 状態ごとに許可される移行イベントの数が、ディテクターモデル定義の##########の制限を超えました。

  解決策: 状態に 20 を超える移行イベントがある場合、このエラーメッセージが表示されることがあります。ディテクターモデルの状態ごとに最大 20 の移行イベントを持つことができます。

- メッセージ: 状態ごとに許可されるイベントの数が、ディテクターモデル定義の#########の制限を超えました。

  解決策: 状態に 20 を超えるイベントがある場合、このエラーメッセージが表示されることがあります。ディテクターモデルの状態ごとに最大 20 のイベントを持つことができます。

- メッセージ: 単一の入力に関連付けることができるディテクターモデルの最大数が制限に達している可能性があります。入力###は、##############ディテクターモデルルートで使用されます。

  解決策: 入力を 10 を超えるディテクターモデルに送信しようとすると、この警告メッセージが表示される場合があります。1 つのディテクターモデルに最大 10 の異なるディテクターモデルを関連付けることができます。

structure

`structure` に関する情報を含む分析結果は、次のエラーメッセージに対応します。

- メッセージ: アクションには 1 つのタイプしか定義されていない可能性がありますが、####タイプのアクションが見つかりました。別のアクションに分割してください。

  解決策: APIオペレーションを使用してディテクターモデルを作成または更新することにより、1 つのフィールドに 2 つ以上のアクションを指定した場合、このエラーメッセージが表示されることがあります。`Action` オブジェクトの配列を定義できます。各アクションを個別のオブジェクトとして定義していることを確認してください。

- メッセージ: TransitionEvent #######は存在しない状態###に移行します。

  解決策： 移行イベントが参照した次の状態 AWS IoT Events が で見つからない場合、このエラーメッセージが表示されることがあります。次の状態が定義されていること、および正しい状態名を入力したことを確認してください。

- メッセージ: DetectorModelDefinition に共有状態名がありました。####の繰り返しで、状態###が見つかりました。

  解決策: 1 つ以上の状態に同じ名前を使用すると、このエラーメッセージが表示される場合があります。ディテクターモデルの各状態に一意の名前を付けるようにしてください。状態名は 1〜128 文字である必要があります。有効な文字: a-z、A-Z、0-9、_ (アンダースコア)、および - (ハイフン)。

- メッセージ: 定義の initialStateName #####が定義された状態に対応していませんでした。

  解決策: 初期状態名が正しくない場合、このエラーメッセージが表示されることがあります。ディテクターモデルは、入力が到着するまで初期 (スタート) 状態のままです。入力が到着すると、ディテクターモデルはすぐに次の状態に移行します。初期状態名が定義された状態の名前であり、正しい名前を入力していることを確認してください。

- メッセージ: ディテクターモデル定義は、条件で少なくとも 1 つの入力を使用する必要があります。

  解決策: 条件で入力を指定しなかった場合、このエラーが発生する可能性があります。少なくとも 1 つの条件で少なくとも 1 つの入力を使用する必要があります。それ以外の場合、 AWS IoT Events は受信データを評価しません。

- メッセージ: SetTimer で設定できるのは秒と durationExpression のいずれか 1 つだけです。

解決策: タイマーに seconds と durationExpression の両方を使用した場合、このエラーメッセージが表示されることがあります。SetTimerAction のパラメータとして seconds または durationExpression のいずれかを使用していることを確認してください。詳細については、「AWS IoT Events API リファレンス」の「SetTimerAction」を参照してください。

- メッセージ: ディテクターモデルのアクションに接続できません。アクションを開始する条件を確認してください。

  解決策: ディテクターモデルのアクションに接続できない場合、イベントの条件は「false」と評価されます。アクションを含むイベントの条件をチェックして、その条件が「true」と評価されていることを確認します。イベントの条件が「true」に評価されていると、アクションは接続可能になります。

- メッセージ: 入力属性が読み込まれていますが、タイマーが切れたことが原因と考えられます。

  解決策: 入力属性の値は、次のいずれかが発生した場合に読み取ることができます。

  - 新しい入力値を受け取ったとき。
  - ディテクターのタイマーが切れたとき。

  入力属性が、その入力の新しい値を受け取ったときにのみ評価されるようにするには、次に示すように条件に triggerType(“Message”) 関数への呼び出しを含めてください。

  ディテクターモデルで評価中の元の条件:

  ```
  if ($input.HeartBeat.status == “OFFLINE”)
  ```

  次に類似したものになります。

  ```
  if ( triggerType("MESSAGE") &&  $input.HeartBeat.status == “OFFLINE”)
  ```

  条件で提供された初期入力の前に triggerType(“Message”) 関数への呼び出しが来る場合。この手法を使用すると、triggerType("Message") 関数は「true」と評価され、新しい入力値を受け取る条件を満たします。triggerType 関数の使用方法の詳細については、「AWS IoT Events 開発者ガイド」の「表現」セクションの triggerType を検索してください。

- メッセージ: ディテクターモデルの状態に到達できません。目的の状態への移行を引き起こす条件を確認してください。

解決策: ディテクターモデル内の状態に到達できない場合、その状態への受信移行を引き起こす条件は「false」と評価されます。ディテクターモデル内のその到達できない状態への受信移行の条件が「true」と評価され、目的の状態に到達できることを確認してください。

- メッセージ: タイマーが切れると、予期しない量のメッセージが送信される可能性があります。

  解決策: タイマーが切れたためにディテクターモデルが予期しない量のメッセージを送信する無限状態になるのを防ぐには、次のようなディテクターモデルの条件で `triggerType("Message")` 関数への呼び出しを使用することを検討してください。

  ディテクターモデルで評価中の元の条件:

  ```
  if (timeout("awake"))
  ```

  次に類似したような条件に変換されます。

  ```
  if (triggerType("MESSAGE") && timeout("awake"))
  ```

  条件で提供された初期入力の前に `triggerType(“Message”)` 関数への呼び出しが来る場合。

  この変更により、ディテクターでタイマーアクションが始まるのを防ぎ、メッセージが無限に送信されるのを防止できます。ディテクターでタイマーアクションを使用する方法の詳細については、{ AWS IoT Events 開発者ガイド」の「ビルトインアクションの使用」ページを参照してください。

expression-syntax

`expression-syntax` に関する情報を含む分析結果は、次のエラーメッセージに対応します。

- メッセージ: ペイロード表現 {##} が無効です。定義されたペイロードタイプは JSON であるため、文字列に対して評価 AWS IoT Events される式を指定する必要があります。

  解決策： 指定されたペイロードタイプが JSON の場合、 AWS IoT Events は最初に、サービスが式を文字列に評価できるかどうかをチェックします。評価結果をブール値または数値にすることはできません。検証が成功しない場合、このエラーが発生する可能性があります。

- メッセージ: `SetVariableAction.value` は表現である必要があります。値 「###」の分析に失敗しました

解決策: `SetVariableAction` を使用して、`name` と `value` で可変を定義できます。`value` は、文字列、数値、またはブール値にすることができます。`value` の表現を指定することもできます。詳細については、AWS IoT Events API リファレンスの SetVariableAction を参照してください。

- メッセージ: DynamoDB アクションの属性 (###) の表現を分析できませんでした。正しい構文で表現を入力してください。

  解決策: `DynamoDBAction` 置換テンプレートのすべてのパラメータに表現を使用する必要があります。詳細については、「AWS IoT Events API リファレンス」の「DynamoDBAction」を参照してください。

- メッセージ: DynamoDBv2 アクションの tableName の表現を分析できませんでした。正しい構文で表現を入力してください。

  解決策: `DynamoDBv2Action` の `tableName` は文字列である必要があります。`tableName` の表現を使用する必要があります。表現は、リテラル、演算子、関数、リファレンス、および置換テンプレートを受け入れます。詳細については、「AWS IoT Events API リファレンス」の「DynamoDBv2Action 」を参照してください。

- メッセージ: 表現を有効な JSON として評価できませんでした。DynamoDBv2 アクションは、JSON ペイロードタイプのみをサポートします。

  解決策: `DynamoDBv2` のペイロードタイプは JSON である必要があります。が有効な JSON へのペイロードのコンテンツ式を評価 AWS IoT Events できることを確認します。詳細については、「AWS IoT Events APIリファレンス」の「DynamoDBv2Action」を参照してください。

- メッセージ: ########のペイロードのコンテンツ表現を分析できませんでした。正しい構文でコンテンツ表現を入力してください。

  解決策: コンテンツ表現には、文字列 (「#####」)、可変 ($variable.###)、入力値 ($input.###.#########)、文字列の連結、および `${}` を含む文字列を含めることができます。

- メッセージ: カスタマイズされたペイロードは空でない必要があります。

  解決策： アクションにカスタムペイロードを選択し、 AWS IoT Events コンソールにコンテンツ式を入力しなかった場合、このエラーメッセージが表示されることがあります。[Custom payload] (カスタムペイロード) を選択した場合は、[Custom payload] (カスタムペイロード) の下にコンテンツ表現を入力する必要があります。詳細については、「AWS IoT Events API リファレンス」の「ペイロード」を参照してください。

- メッセージ: タイマー「#####」持続時間の表現「#######」の分析に失敗しました。

解決策: タイマー持続時間の表現の評価結果は、60〜31,622,400 の値である必要があります。期間の評価結果は、最も近い整数に切り捨てられます。

- メッセージ: ######の表現「##」の分析に失敗しました

  解決策: 指定されたアクションの表現の構文が正しくない場合、このメッセージが表示されることがあります。正しい構文で表現を入力していることを確認してください。詳細については、「デバイスデータをフィルタリングし、 でアクションを定義する構文 AWS IoT Events」を参照してください。

- メッセージ: IotSitewiseAction の *fieldName* を分析できませんでした。表現では正しい構文を使用する必要があります。

  解決策： が の *fieldName* を解析できなかった場合 AWS IoT Events 、このエラーが表示されることがありますIotSitewiseAction。*fieldName* が AWS IoT Events が分析できる表現を使用していることを確認してください。詳細については、「AWS IoT Events API リファレンス」の「IotSiteWiseAction」を参照してください。

data-type

data-type に関する情報を含む分析結果は、次のエラーメッセージに対応します。

- メッセージ: タイマー#####持続時間の表現#######が無効です。数値を返す必要があります。

  解決策： AWS IoT Events がタイマーの期間式を数値に評価できなかった場合、このエラーメッセージが表示されることがあります。durationExpression を数値に変換できることを確認してください。ブール値などの他のデータ型はサポートされていません。

- メッセージ: 表現 *condition-expression* は有効な条件表現ではありません。

  解決策： AWS IoT Events が をブール値condition-expressionに評価できなかった場合、このエラーメッセージが表示されることがあります。ブール値は、TRUE か FALSE のいずれかである必要があります。条件表現ブール値に変換できることを確認してください。結果がブール値でない場合、FALSE と同等であり、アクションを呼び出したり、イベントで指定された nextState に移行したりしません。

- メッセージ:互換性のないデータ型[###]が次の表現で######用に見つかりました。##

  解決策: 解決策: ディテクターモデルの同じ入力属性または可変のすべての表現は、同じデータ型をリファレンスする必要があります。

次の情報を使用して、問題を解決してください。

- 1 つ以上の演算子でオペランドとしてリファレンスを使用する場合は、リファレンスするすべてのデータ型に互換性があることを確認してください。

  例えば、次の表現では、整数 2 は == 演算子と && 演算子の両方のオペランドです。オペランドに互換性があることを確認するには、`$variable.testVariable + 1` と `$variable.testVariable` は整数または 10 進数をリファレンスする必要があります。

  また、整数 1 は + 演算子のオペランドです。したがって、`$variable.testVariable` は整数または 10 進数をリファレンスする必要があります。

  ```
  '$variable.testVariable + 1 == 2 && $variable.testVariable'
  ```

- 関数に渡される引数としてリファレンスを使用する場合は、関数が参照するデータ型をサポートしていることを確認してください。

  例えば、次の `timeout("time-name")` 関数では、引数として二重引用符を含む文字列が必要です。*timer-name* 値のリファレンスを使用する場合は、二重引用符で囲まれた文字列を参照する必要があります。

  ```
  timeout("timer-name")
  ```

  > **Note**
  >
  > `convert(type, expression)` 関数の場合、*###*値にリファレンスを使用する場合、リファレンスの評価結果は String、Decimal、または Boolean である必要があります。

  詳細については、「AWS IoT Events 式の入力と変数の リファレンス」を参照してください。

- メッセージ: *######*で使用される互換性のないデータ型 [*###*]。これにより、ランタイムエラーが発生する可能性があります。

  解決策: 同じ入力属性または可変の 2 つの表現が 2 つのデータ型をリファレンスしている場合、この警告メッセージが表示されることがあります。同じ入力属性または可変の表現が、ディテクターモデルの同じデータ型をリファレンスしていることを確認してください。

- メッセージ: 演算子 [*operator*] に入力したデータ型 [*inferred-types*] は、次の表現と互換性がありません。「*##*」

解決策: 表現が、指定された演算子と互換性のないデータ型を組み合わせている場合、このエラーメッセージが表示されることがあります。例えば、次の表現では、演算子 + は整数、10 進、および文字列のデータ型と互換性がありますが、ブールデータ型のオペランドとは互換性がありません。

```
true + false
```

演算子で使用するデータ型に互換性があることを確認する必要があります。

- メッセージ: #### で見つかったデータ [#####] には互換性がなく、ランタイムエラーが発生する可能性があります。

  解決策: 同じ入力属性の 2 つの表現が、状態の `OnEnterLifecycle`、または状態の `OnInputLifecycle` と `OnExitLifecycle` の両方の2つのデータ型をリファレンスしている場合に、このエラーメッセージが表示されることがあります。`OnEnterLifecycle` (または `OnInputLifecycle` と `OnExitLifecycle` の両方) の表現が、ディテクターモデルの各状態に対して同じデータ型をリファレンスしていることを確認してください。

- メッセージ: ペイロード表現 [##] が無効です。ペイロードタイプは JSON 形式であるため、ランタイムに文字列に評価される表現を指定します。

  解決策： 指定したペイロードタイプが JSON であっても、その式を文字列に評価 AWS IoT Events できない場合、このエラーが表示されることがあります。評価結果がブール値や数値ではなく、文字列であることを確認してください。

- メッセージ: 補間された表現 {####} は、ランタイムに整数またはブール値のいずれかに評価される必要があります。そうしないと、ペイロード表現 {######} はランタイムに有効な JSON として分析できません。

  解決策： 補間された式を整数またはブール値に評価 AWS IoT Events できなかった場合、このエラーメッセージが表示されることがあります。tringなどの他のデータ型はサポートされていないため、補間された表現を整数またはブール値に変換できることを確認してください。

- メッセージ: `IotSitewiseAction` フィールド##の表現タイプは、#####タイプとして定義され、#####タイプとして推測されます。定義されたタイプと推測されたタイプは同じである必要があります。

  解決策: `IotSitewiseAction` の `propertyValue` の表現に、 AWS IoT Eventsによって推測されたデータ型とは異なる方法で定義されたデータ型がある場合、このエラーメッセージが表示される

ことがあります。ディテクターモデルのこの表現のすべてのインスタンスに同じデータ型を使用していることを確認してください。

- メッセージ: `setTimer` アクションに使用されるデータ型 [###] は、次の表現の `Integer` に評価されません。##

  解決策: 期間表現の推定データ型が整数または小数でない場合、このエラーメッセージが表示されることがあります。`durationExpression` を数値に変換できることを確認してください。ブール値や文字列などの他のデータ型はサポートされていません。

- メッセージ: 比較演算子 [######] のオペランドで使用されるデータ型 [#####] は、次の表現で互換性がありません: ##

  解決策: ディテクターモデルの条件表現 (##) の演算子の#####の推定データ型が一致しません。オペランドは、ディテクターモデルのすべての 他の パートで一致するデータ型で使用する必要があります。

> **Tip**
>
> `convert` を使用して、ディテクターモデルの表現のデータ型を変更できます。詳細については、「AWS IoT Events 式で使用する関数」を参照してください。

`referenced-data`

`referenced-data` に関する情報を含む分析結果は、次のエラーメッセージに対応します。

- メッセージ: 壊れたタイマーが検出されました: タイマー#####が表現で使用されていますが、設定されていません。

  解決策: 設定されていないタイマーを使用すると、このエラーメッセージが表示される場合があります。表現で使用する前に、タイマーを設定する必要があります。また、正しいタイマー名を入力していることを確認してください。

- メッセージ: 壊れた可変が検出されました: 可変###が表現で使用されていますが、設定されていません。

  解決策: 設定されていない可変を使用すると、このエラーメッセージが表示される場合があります。表現で使用する前に、可変を設定する必要があります。また、正しい可変名を入力していることを確認してください。

- メッセージ: 壊れた可変が検出されました: 可変に値を設定する前に、表現で使用されています。

解決策: 表現で評価できるには、各可変に値に割り当てられている必要があります。値を取得できるように、毎回使用する前に可変の値を設定してください。また、正しい可変名を入力していることを確認してください。

`referenced-resource`

`referenced-resource` に関する情報を含む分析結果は、次のエラーメッセージに対応します。

- メッセージ: ディテクターモデル定義に、存在しない入力へのリファレンスが含まれています。

  解決策:表現を使用して存在しない入力をリファレンスすると、このエラーメッセージが表示される場合があります。表現が既存の入力をリファレンスしていることを確認し、正しい入力名を入力してください。入力がない場合は、最初に入力を作成します。
- メッセージ: ディテクターモデルの定義 に無効な InputName が含まれています: ###

  解決策: ディテクターモデルに無効な入力名が含まれている場合、このエラーメッセージが表示されることがあります。正しい入力名を入力してください。入力名は 1〜128 文字である必要があります。有効な文字: a-z、A-Z、0-9、_ (アンダースコア)、および - (ハイフン)。

のディテクターモデルを分析する AWS IoT Events（コンソール）

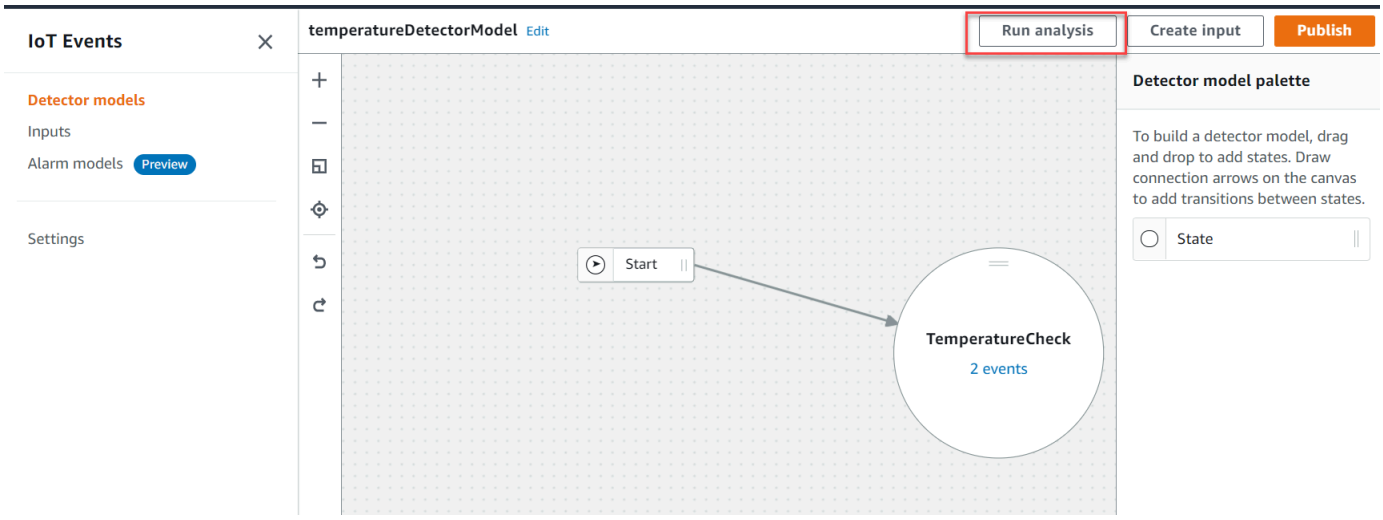
AWS IoT Events では、 AWS IoT Events API を使用してイベントを検出し、アクションをトリガーすることで、IoT データを監視して対応できます。次の手順では、 AWS IoT Events コンソールを使用してディテクターモデルを分析します。

> **Note**
>
> がディテクターモデルの分析 AWS IoT Events を開始した後、分析結果を取得するまでに最大 24 時間かかります。

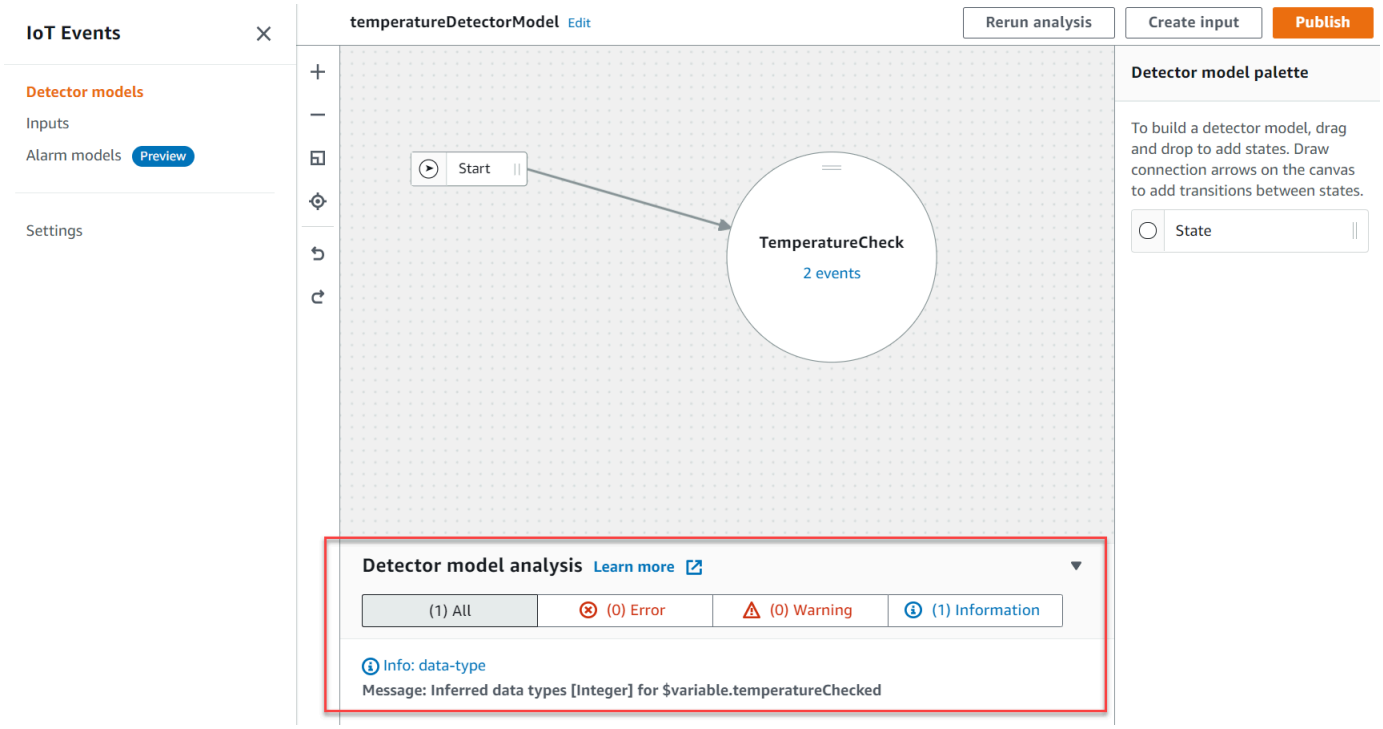
ディテクターモデル分析は、モデルを最適化し、潜在的な問題を特定し、意図したとおりに機能していることを確認するのに役立ちます。例えば、風力発電所では、ディテクターモデル分析により、異常な振動パターンに基づいてモデルが潜在的な歯車の障害を正しく識別するかどうかが明らかになる可能性があります。または、風速が安全な動作しきい値を超えたときにモデルがメンテナンスアラートを正確にトリガーする場合。分析に基づいてモデルを改良することで、予測メンテナンスを改善し、ダウンタイムを減らし、全体的なエネルギー生産効率を向上させることができます。

ディテクターモデルを分析するには

1. AWS IoT Events コンソール にサインインします。
2. ナビゲーションペインで、ディテクターモデルを選択します。
3. ディテクターモデルで、ターゲットディテクターモデルを選択します。
4. ディテクターモデルのページで、編集を選択します。
5. 右上隅で、分析の実行を選択します。



AWS IoT Events コンソールの分析結果の例を次に示します。



AWS IoT Events （ AWS CLI) でディテクターモデルを分析する

AWS IoT Events ディテクターモデルをプログラムで分析することで、その構造、動作、パフォーマンスに関する貴重なインサイトが得られます。この API ベースのアプローチにより、自動分析、既存のワークフローとの統合、複数のディテクターモデルで一括操作を実行できるようになります。StartDetectorModelAnalysis API を活用することで、モデルの詳細な調査を開始できるため、潜在的な問題を特定し、ロジックフローを最適化し、IoT イベント処理がビジネス要件と一致していることを確認するのに役立ちます。

次の手順では、 AWS CLI を使用してディテクターモデルを分析します。

を使用してディテクターモデルを分析するには AWS CLI

1. 次のコマンドを実行して、分析をスタートします。

```
aws iotevents start-detector-model-analysis --cli-input-json file://file-name.json
```

> **Note**
>
> *#####*を、ディテクターモデル定義を含むファイルの名前に置き換えます。

Example ディテクターモデルの定義

```
{
    "detectorModelDefinition": {
        "states": [
            {
                "stateName": "TemperatureCheck",
                "onInput": {
                    "events": [
                        {
                            "eventName": "Temperature Received",
                            "condition":
 "isNull($input.TemperatureInput.sensorData.temperature)==false",
                            "actions": [
                                {
                                    "iotTopicPublish": {
                                        "mqttTopic": "IoTEvents/Output"
                                    }
                                }
```

```
                            ]
                        }
                    ],
                    "transitionEvents": []
                },
                "onEnter": {
                    "events": [
                        {
                            "eventName": "Init",
                            "condition": "true",
                            "actions": [
                                {
                                    "setVariable": {
                                        "variableName": "temperatureChecked",
                                        "value": "0"
                                    }
                                }
                            ]
                        }
                    ]
                },
                "onExit": {
                    "events": []
                }
            }
        ],
        "initialStateName": "TemperatureCheck"
    }
}
```

を使用して既存のディテクターモデル AWS CLI を分析する場合は、次のいずれかを選択してディテクターモデル定義を取得します。

- AWS IoT Events コンソールを使用する場合は、次の操作を行います。
  1. ナビゲーションペインで、ディテクターモデルを選択します。
  2. ディテクターモデルで、ターゲットディテクターモデルを選択します。
  3. アクションからディテクターモデルをエクスポートを選択して、ディテクターモデルをダウンロードします。ディテクターモデルは JSON で保存されます。
  4. ディテクターモデルの JSON ファイルを開きます。
  5. 必要なのは `detectorModelDefinition` オブジェクトだけです。以下を削除します。

- ページ上部の最初の中括弧 ({)
- `detectorModel` 線
- `detectorModelConfiguration` オブジェクト
- ページ下部の最後の中括弧 (})

6. ファイルを保存します。

- を使用する場合は AWS CLI、次の操作を行います。

1. ターミナルで次のコマンドを実行します。

```
aws iotevents describe-detector-model --detector-model-name detector-model-name
```

2. ############をディテクターモデルの名前に置き換えます。
3. `detectorModelDefinition` オブジェクトをテキストエディタにコピーします。
4. `detectorModelDefinition` の外側に中括弧 ({}) を追加します。
5. ファイルを JSON で保存します。

Example レスポンスの例

```
{
    "analysisId": "c1133390-14e3-4204-9a66-31efd92a4fed"
}
```

2. 出力から分析 ID をコピーします。
3. 次のコマンドを実行して、分析のステータスを取得します。

```
aws iotevents describe-detector-model-analysis --analysis-id "analysis-id"
```

> **Note**
>
> *analysis-id* をコピーした分析 ID に置き換えます。

Example レスポンスの例

```
{
    "status": "COMPLETE"
```

```
}
```

ステータスは、次のいずれかの値になります。

- RUNNING – AWS IoT Events はディテクターモデルを分析しています。このプロセスは、完了するまでに最大 1 分かかる場合があります。
- COMPLETE – ディテクターモデルの分析 AWS IoT Events を完了しました。
- FAILED - AWS IoT Events ディテクターモデルを分析できませんでした。あとでもう一度試してみてください。

4. 次のコマンドを実行して、ディテクターモデルの 1 つ以上の分析結果を取得します。

> **Note**
>
> *analysis-id* をコピーした分析 ID に置き換えます。

```
aws iotevents get-detector-model-analysis-results --analysis-id "analysis-id"
```

Example レスポンスの例

```
{
    "analysisResults": [
        {
            "type": "data-type",
            "level": "INFO",
            "message": "Inferred data types [Integer] for
 $variable.temperatureChecked",
            "locations": []
        },
        {
            "type": "referenced-resource",
            "level": "ERROR",
            "message": "Detector Model Definition contains reference to Input
 'TemperatureInput' that does not exist.",
            "locations": [
                {
                    "path": "states[0].onInput.events[0]"
                }
            ]
```

```
        }
    ]
}
```

> **Note**
>
> がディテクターモデルの分析 AWS IoT Events を開始した後、分析結果を取得するまでに最大 24 時間かかります。

AWS IoT Events コマンド

この章では、 で使用できるすべての API オペレーションの包括的なガイドを提供します AWS IoT Events。サポートされているウェブサービスプロトコル全体で、各オペレーションのサンプルリクエスト、レスポンス、潜在的なエラーなど、詳細な説明を提供します。これらの API オペレーションを理解することで、IoT アプリケーション AWS IoT Events に効果的に統合し、イベント検出とレスポンスのワークフローを自動化できます。

AWS IoT Events アクション

AWS IoT Events API コマンドを使用して、入力モデルとディテクターモデルを作成、読み取り、更新、削除し、それらのバージョンを一覧表示できます。詳細については、 AWS IoT Events API リファレンスの AWS IoT Events でサポートされているアクションとデータ型を参照してください。

AWS CLI コマンドリファレンスAWS IoT Events のセクションには、管理および操作に使用できる AWS CLI コマンドが含まれています AWS IoT Events。

AWS IoT Events データ

AWS IoT Events Data API コマンドを使用して、ディテクターへの入力の送信、ディテクターの一覧表示、ディテクターのステータスの表示または更新を行うことができます。詳細については、AWS IoT Events 「 API リファレンス」の AWS IoT Events 「Data」でサポートされているアクションとデータ型を参照してください。

AWS CLI コマンドリファレンスAWS IoT Events のデータセクションには、 AWS IoT Events データの処理に使用できる AWS CLI コマンドが含まれています。

のドキュメント履歴 AWS IoT Events

次の表は、2020 年 9 月 17 日より後のの AWS IoT Events デベロッパーガイドの重要な変更点をまとめたものです。このドキュメントの更新情報は、RSS フィードに加入して取得できます。

変更	説明	日付
サポート終了通知	サポート終了通知: 2026 年 5 月 20 日、 AWS はサポートを終了します AWS IoT Events。2026 年 5 月 20 日以降、 AWS IoT Events コンソールまたは AWS IoT Events リソースにアクセスできなくなります。	2025 年 5 月 20 日
リージョンへの参入	AWS IoT Events がアジアパシフィック (ムンバイ) リージョンで利用可能になりました。	2021 年 9 月 30 日
リージョン起動	AWS IoT Events が AWS GovCloud (米国西部) リージョンで利用可能になりました。	2021 年 9 月 22 日
分析を実行してディテクターモデルをトラブルシューティングする	AWS IoT Events でディテクターモデルを分析し、ディテクターモデルのトラブルシューティングに使用できる分析結果を生成できるようになりました。	2021 年 2 月 23 日
リージョン起動	中国 (北京) AWS IoT Events で開始されました。	2020 年 9 月 30 日
表現使用法	表現の書き込みを示す例を追加しました。	2020 年 9 月 22 日

アラームによるモニタリング	アラームは、データの変更をモニタリングするのに役立ちます。しきい値に違反したときに通知を送信するアラームを作成できます。	2020 年 6 月 1 日

以前の更新

次の表は、2020 年 9 月 18 日より前のAWS IoT Events デベロッパーガイドの重要な変更点をを示しています。

変更	説明	日付
式リファレンスに型検証を追加	式リファレンスに型検証情報を追加しました。	2020 年 8 月 3 日
他のサービスのリージョン警告を追加	AWS IoT Events および他の AWS のサービスで同じリージョンを選択することに関する警告を追加しました。	2020 年 5 月 7 日
追加、更新	• ペイロードのカスタマイズ機能 • 新しいイベントアクション: Amazon DynamoDB と AWS IoT SiteWise	2020 年 4 月 27 日
ディテクターモデル条件式用の組み込み関数を追加	ディテクターモデル条件式用の組み込み関数を追加しました。	2019 年 9 月 10 日
ディテクターモデルの例を追加	ディテクターモデルの例を追加しました。	2019 年 8 月 5 日
新しいイベントアクションを追加	次の新しいイベントアクションを追加しました。	2019 年 7 月 19 日

変更	説明	日付
	• Lambda • Amazon SQS • Kinesis Data Firehose • AWS IoT Events 入力	
追加、修正	• `timeout()` 関数の説明を更新しました。 • 休止状態アカウントに関するベストプラクティスを追加。	2019 年 6 月 11 日
アクセス許可ポリシーとコンソールのデバッグオプションを更新	• コンソールのアクセス許可ポリシーを更新しました。 • コンソールのデバッグオプションページイメージを更新しました。	2019 年 6 月 5 日
更新	AWS IoT Events サービスは一般公開されています。	2019 年 5 月 30 日
追加、更新	• セキュリティ情報を更新しました。 • 注釈付きディテクターモデルの例を追加しました。	2019 年 5 月 22 日
例と必要なアクセス許可を追加	Amazon SNS ペイロードの例を追加しました。 に必要なアクセス許可が追加されました `CreateDetectorModel` 。	2019 年 5 月 17 日
追加のセキュリティ情報を追加	セキュリティセクションに情報を追加しました。	2019 年 5 月 9 日
限定プレビューリリース	ドキュメントの限定プレビューリリース。	2019 年 3 月 28 日