



開発者ガイド

AWS IoT Events



AWS IoT Events: 開発者ガイド

Copyright © 2024 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 が所有していない他のすべての商標は、それぞれの所有者の所有物であり、Amazon と提携、接続、または後援されている場合とされていない場合があります。

Table of Contents

とは AWS IoT Events?	1
利点と特徴	1
ユースケース	2
設定	4
のアクセス許可の設定 AWS IoT Events	4
アクション許可	5
入力データの保護	7
Amazon CloudWatch ログ記録ロールポリシー	8
Amazon SNS メッセージングロールポリシー	10
開始	11
前提条件	13
入力の作成	14
ナビゲーションペインへの入力の作成	15
ディテクターモデルへの入力の作成	15
ディテクターモデルの作成	15
ディテクターモデルをテストするための入力を送信します	23
ベストプラクティス	27
AWS IoT Events ディテクターモデルの開発時に Amazon CloudWatch ログを有効にする	27
コンソールでの作業中は、定期的に公開してディテクターモデルを保存してください。AWS IoT Events	28
長時間使用しないことによるデータ損失を防ぐため、データを保存してください AWS IoT Events 。	28
チュートリアル	29
AWS IoT Events を使用して IoT デバイスをモニタリングする	29
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？	31
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？	32
簡単なステップバイステップの例	33
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	35
デバイスの状態を表すディテクターモデルを作成する	36
ディテクターへの入力としてメッセージを送信する	39
ディテクターモデルの制限と制限	43
コメント例: HVAC 温度制御	46

背景	46
サポートされているアクション	83
組み込みアクションの使用	84
タイマーアクションの設定	84
タイマーアクションのリセット	84
タイマーアクションをクリア	85
可変アクションの設定	85
他の AWS サービスの使用	86
AWS IoT Core	87
AWS IoT Events	88
AWS IoT SiteWise	89
Amazon DynamoDB	91
Amazon DynamoDB (v2)	94
Amazon Data Firehose	95
AWS Lambda	96
Amazon Simple Notification Service	97
Amazon Simple Queue Service	98
表現	100
構文	100
リテラル	100
演算子	100
関数	102
リファレンス	106
置換テンプレート	109
使用方法	109
AWS IoT Events 表現を書き込み	110
ディテクターモデルの例	112
HVAC (空気調和工学) の温度制御	112
背景	112
入力定義	113
ディテクターモデルの定義	115
BatchPutMessage の例	132
BatchUpdateDetector の例	138
AWS IoT Core ルールエンジンの例	140
クレーン	143
背景	143

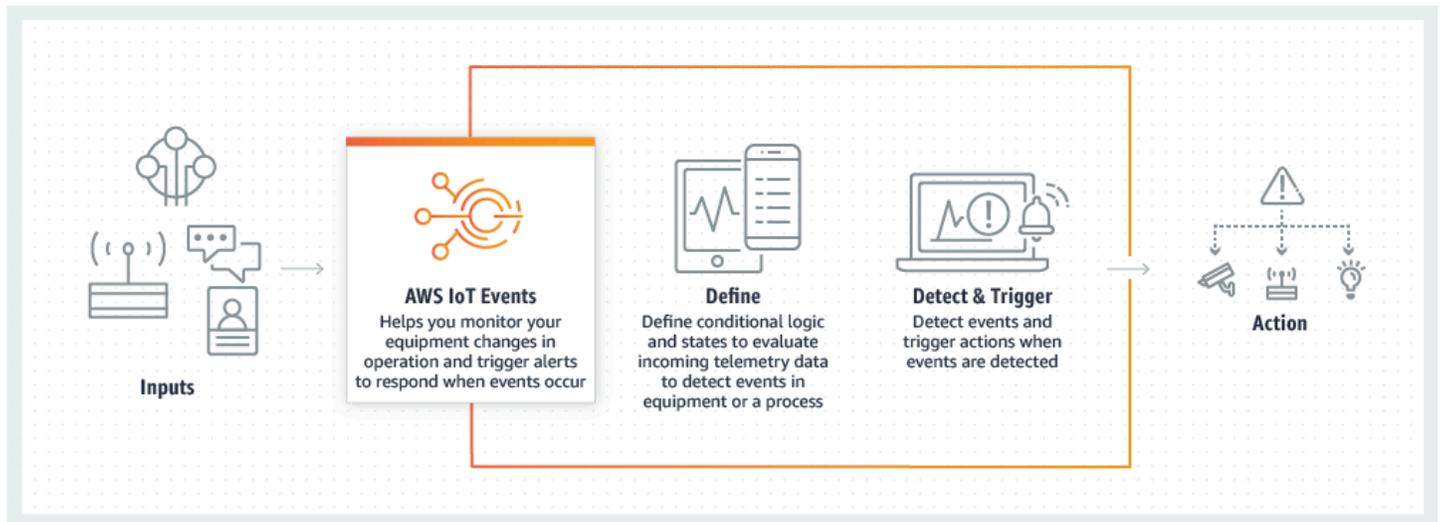
コマンド	144
ディテクターモデル	145
入力	152
メッセージ	152
センサーとアプリケーションによるイベント検出	154
デバイス HeartBeat	156
ISA アラーム	158
シンプルなアラーム	168
アラームによるモニタリング	173
AWS IoT SiteWise での作業	173
フローの確認	173
アラームモデルの作成	174
要件	175
アラームモデルの作成 (コンソール)	175
アラームへの対応	178
アラームへの対応 (コンソール)	178
アラームへの対応 (API)	179
アラーム通知の管理	179
Lambda 関数の作成	179
AWS IoT Events が提供する Lambda 関数を使用する	188
受信者の管理	189
セキュリティ	191
Identity and access management	191
対象者	192
アイデンティティを使用した認証	193
ポリシーを使用したアクセス権の管理	196
詳細はこちら	198
AWS IoT Events と IAM の連携方法	198
アイデンティティベースのポリシーの例	202
サービス間の混乱した代理の防止	208
トラブルシューティング	212
モニタリング	214
モニタリングツール	215
Amazon CloudWatch でのモニタリング	216
AWS CloudTrail を使用した AWS IoT Events API呼び出しのロギング	217
コンプライアンス検証	235

耐障害性	236
インフラストラクチャセキュリティ	236
クォータ	238
タグ付け	239
ベーシックタグ	239
タグの制約と制限	240
IAM ポリシーでのタグの使用	240
トラブルシューティング	244
一般的な AWS IoT Events 問題と解決策	244
ディテクターモデルの作成エラー	244
削除されたディテクターモデルからの更新	245
アクショントリガーの障害 (条件を満たす場合)	245
アクショントリガーの障害 (しきい値を超えた場合)	245
誤った状態の使用法	246
接続メッセージ	246
InvalidRequestException メッセージ	246
Amazon CloudWatch Logs action.setTimer エラー	247
Amazon CloudWatch ペイロードエラー	248
互換性のないデータ型	249
へのメッセージの送信に失敗しました AWS IoT Events	250
ディテクターモデルのトラブルシューティング	251
診断情報	252
ディテクターモデルの分析 (コンソール)	265
ディテクターモデルの分析 (AWS CLI)	266
コマンド	271
AWS IoT Events アクション	271
AWS IoT Events データ	271
ドキュメント履歴	272
以前の更新	273
.....	cclxxv

とは何ですか AWS IoT Events?

AWS IoT Events 機器やデバイス群に障害や動作の変化がないかを監視し、そのような事象が発生したときにアクションを起こすことができます。AWS IoT Events デバイス、プロセス、アプリケーション、AWS その他のサービスからの IoT センサーデータを継続的に監視して、重要なイベントを特定し、対策を講じることができます。

を使用すると AWS IoT Events 、 AWS IoT Events コンソールや API AWS からアクセスできる複雑なイベント監視アプリケーションをクラウドに構築できます。



利点と特徴

複数の出典からの入力を受け入れる

AWS IoT Events 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。これらには、センサーデバイス、管理アプリケーション、AWS IoT およびその他のサービス (AWS IoT Core AWS IoT Analyticsやなど) が含まれます。標準の API インターフェイス (BatchPutMessageAPI) を使用して、AWS IoT Events 任意のテレメトリデータ入力をにプッシュできます。

単純な論理的な表現を使用して複雑なイベントのパターンを認識する

AWS IoT Events 単一の IoT デバイスやアプリケーション、またはさまざま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。AWS IoT Events 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ディテクターを構成できます。

イベントに基づき、アクションのトリガーを行う

AWS IoT Events Amazon Simple Notification Service (Amazon SNS)、Lambda、Amazon SQS、AWS IoT Core、Amazon Kinesis Firehose でアクションを直接トリガーできます。AWS Lambda、AWS IoT ルールエンジンを使用して関数を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なります。

AWS IoT Events には実行可能なアクションのライブラリが事前に用意されており、独自のアクションを定義することもできます。

フリートの需要に合わせてオートスケーリングする

AWS IoT Events 同種のデバイスを接続すると自動的にスケーリングされます。特定のタイプのデバイスに対してディテクタを一度定義すると、接続先のデバイスのすべてのインスタンスがサービスによって自動的にスケーリングされ、管理されます。AWS IoT Events

ユースケース

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をモニタリングする必要があります。機能が停止し、障害の原因について追加のコンテキストがないならば、プロセッシング単位または機械全体の迅速な交換が必要な場合があります。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、時間の経過とともに送信されるエラーコードを使用して問題を正確に診断できます。すべてを交換する代わりに、交換が必要な部分のみを含む技術者に送信するために必要な情報が得られます。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。

産業用ロボットの管理

施設内にロボットをデプロイして、パッケージの移動を自動化します。ロボットのコストを最小限に抑えるために、ロボットはクラウドに情報を報告するシンプルで低コストのセンサーを備えています。しかし、ロボットには数十のセンサーと何百ものオペレーションモードがあり、問題の発生時に検出するのが難しくなります。を使用すると AWS IoT Events、センサーデータをクラウドで処理し、障害が差し迫っている場合に技術スタッフに自動的に警告するアラートを作成するエキスパートシステムを構築できます。

構築オートメーションシステムの追跡

これらの環境しきい値に違反したときに発生する機器の障害を防ぐために、高温および低湿度をモニタリングする必要があるラージ数のデータセンターを運用しています。使用するセンサーは多くのメーカーから購入されており、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、さまざまなベンダーの管理ソフトウェアには互換性がないため、問題の検出が困難です。を使用すると AWS IoT Events、故障が発生するかなり前に、冷暖房システムの問題を運用アナリストに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。

セットアップ AWS IoT Events

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、通話呼び出しを受け取り、電話キーパッドで検証コードを入力するように求められます。

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。セキュリティのベストプラクティスとして、ユーザーに管理アクセスを割り当て、ルートユーザーのみを使用して [ルートユーザーアクセスが必要なタスク](#) を実行してください。

のアクセス許可の設定 AWS IoT Events

このセクションでは、AWS IoT Eventsの一部の特徴を使用するために必要なロールと許可について説明します。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、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 リソースにアクセスするため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。に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: -, _, :

アクション許可

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を AWS IoT Events 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 サービスで作業できる以下のアクションをサポートしています。

- [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。

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>:*"
  }
]
}
```

入力データの保護

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。

Amazon CloudWatch ログ記録ロールポリシー

次のポリシードキュメントは、が AWS IoT Events CloudWatch ユーザーに代わって 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があります。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 ログのリソースポリシーを配置できます。これにより、AWS IoT Events はログイベントを CloudWatch ストリームに入れることができます。

```
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 } }"
```

Amazon SNS メッセージングロールポリシー

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、他のユーザーがディテクターモデルを書き込んだディテクターモデルをインポートしたり、別の AWS リージョンでそれらをエクスポート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。

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、他の人が作成したディテクターモデルをインポートしたり、自分のディテクターモデルをエクスポートして、さまざまな 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、への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 に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、データセットの内容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にアクションとして 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に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、への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、[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

- [前提条件](#)
- [入力の作成](#)
- [ディテクターモデルの作成](#)
- [ディテクターモデルをテストするための入力を送信します](#)

前提条件

AWS アカウントをお持ちでない場合は、アカウントを作成します。

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、通話呼び出しを受け取り、電話キーパッドで検証コードを入力するように求められます。

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。セキュリティのベストプラクティスとして、ユーザーに管理アクセスを割り当て、ルートユーザーのみを使用して[ルートユーザーアクセスが必要なタスク](#)を実行してください。

3. 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る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。

4. AWS リージョン をサポートする 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。ヘルプが必要な場合は、AWS Management Console 入門の [AWS Management Console の操作](#) を参照してください。

入力の作成

モデルの入力を構築するとき、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集め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、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。

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
  }
}
```

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。ナビゲーションペインまたはディテクターモデルを使用して入力を作成する方法については、このセクションのトピックのいずれかを使用してください。

トピック

- [ナビゲーションペインへの入力の作成](#)
- [ディテクターモデルへの入力の作成](#)

ナビゲーションペインへの入力の作成

このトピックでは、ナビゲーションペインを使用して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を説明します。

1. [AWS IoT Events コンソールにログインするか](#)、AWS IoT Events 新しい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にある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. 入力には、オプションで「説明」を入力し、「ファイルをアップロード」を選択します。InputName表示されるダイアログボックスで、[\[入力の作成\]](#) の概要で作成した input.json ファイルを選択します。
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] を選択しています。

ディテクターモデルへの入力の作成

このトピックでは、テレメトリデータまたはメッセージを受信するディテクターモデルの入力を定義する方法を説明します。

1. [AWS IoT Events コンソール](#)を開きます。
2. AWS IoT Events コンソールで、[検出器モデルの作成] 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、オプションの説明を入力しInputName、[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、[\[入力の作成\]](#) の概要で作成した input.json ファイルを選択します。
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] を選択しています。

ディテクターモデルの作成

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、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. [保存] 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. [保存] 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. [保存] を選択します。
19. 例にイベントを追加し続けます。
 - a. でOnInput、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. でOnInput、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. でOnExit、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、[ディテクターモデルの制限と制限](#) 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":
"pressureThresholdBreach",
                    "value":
"$variable.pressureThresholdBreach + 3"
                  }
                ],
              "condition": "$input.PressureInput.sensorData.pressure
> 70",
              "nextState": "Dangerous"
            }
          ],
          "events": []
        },
        "stateName": "Normal",
        "onEnter": {
          "events": [
            {
              "eventName": "init",
              "actions": [
                {
                  "setVariable": {
                    "variableName":
"pressureThresholdBreach",
                    "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"
}
}

```

```
}  
}
```

ディテクターモデルをテストするための入力を送信します

でテレメトリデータを受信するには、いくつかの方法があります AWS IoT Events (「」を参照[サポートされているアクション](#))。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、デバイスがメッセージブローカーを使用して MQTT AWS IoT 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。
 - [次へ] をクリックします。
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'** と入力します。

[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 と入力します。
- 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。

- [次へ] をクリックします。
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。

- の手順に従って[簡単なステップバイステップの例](#)、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 [表現](#) の詳細を学びます。
- [サポートされているアクション](#) についてはこちら。
- 何かが機能していない場合は、[AWS IoT Events のトラブルシューティング](#) 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 デテクターモデルの開発時に Amazon CloudWatch ロギングを有効にする](#)
- [コンソールでの作業中は、定期的に公開してデテクタモデルを保存してください。AWS IoT Events](#)
- [長時間使用しないことによるデータ損失を防ぐため、データを保存してください AWS IoT Events。](#)

AWS IoT Events デテクターモデルの開発時に Amazon CloudWatch ロギングを有効にする

Amazon は、CloudWatch AWS AWS お客様のリソースとお客様が実行するアプリケーションをリアルタイムで監視します。これにより CloudWatch、リソースの使用状況、アプリケーションのパフォーマンス、運用状態をシステム全体で可視化できます。AWS IoT Events 検出器モデルを開発またはデバッグすると、AWS IoT Events 何が起きているのか、CloudWatch検出器モデルで発生したエラーを把握するのに役立ちます。

有効にするには CloudWatch

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 AWS IoT Eventsのログを作成および管理する権限を付与するポリシーを添付したルールを作成してください。
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. [ログオプションの編集] ページの [ログオプション] セクションで、次の操作を行います。
 - a. 「詳細レベル」では、オプションを選択します。
 - b. [Select role] (ルールを選択) 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ルールを選択します。

- c. (オプション) 冗長度レベルで「デバッグ」を選択した場合、次のように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「モデルオプションを追加」を選択します。
 - ii. 検出器モデル名と (オプション) を入力して、KeyValue記録する検出器モデルと特定の検出器 (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。

コンソールでの作業中は、定期的に公開してディテクタモデルを保存してください。AWS IoT Events

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ーのローカルに保存されます。ただし、ディテクタモデルを 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。

長時間使用しないことによるデータ損失を防ぐため、データを保存してください AWS IoT Events。

AWS IoT Events 長期間使用しないと、検出器モデルを含むデータが自動的に削除されることがあります。かなりの期間は、例えば、料金が発生せず、ディテクターモデルを作成しないことを意味する場合があります。ただし、少なくとも 30 日前に通知することなく、データまたはディテクターモデルを削除することはありません。長期間データを保存する必要がある場合は、[AWS ストレージサービスの利用](#)をお勧めします。

チュートリアル

この章では、次の方法について説明します。

- デテクターモデルに含める状態を決定し、1つのデテクターインスタスが必要か複数のデテクターインスタ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テクターモデルを作成します。
- 入力、デテクターモデル、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 デバイスをモニタリングする](#)
- [簡単なステップバイステップの例](#)
- [デテクターモデルの制限と制限](#)
- [コメント例: HVAC 温度制御](#)

AWS IoT Events を使用して IoT デバイスをモニタリングする

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。
- AWS IoT Analytics で、[CreateDataset](#) オペレーションを使用して、`contentDeliveryRules` でデータセットを作成します。これらのルールは、データセットの内容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。AWS IoT Events は、多くの出典から JSON ペイロードの形式で入力を受け取ります。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。

デテクターモデルを作成する

状態を使用して、デテクターモデル (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入ったり出た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)、デテクターが作成されます。(各デテクターは、デテクターモデルのインスタンスです。) 次に、新しいデテクターは、デ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テクター (インスタンス) が作成されます。

デ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 AWS IoT

Events に転送する AWS IoT Core ルールエンジンの [AWS IoT Events アクション](#) ルール書き込みします。これを行うには、入力を名前でも識別します。次に、AWS IoT コンソールを引き続き使用して、AWS IoT Events への入力として転送されるいくつかのメッセージを生成します。

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、最初に行うことが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。別の入力に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が。

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括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、その場所の天気か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。

簡単なステップバイステップの例

この例では、AWS IoT Events コマンドを使用して AWS CLI API を呼び出し、エンジンの 2 つの状態 (通常状態と過圧状態) をモデル化するディテクターを作成します。

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。これは、AWS IoT Events への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。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 AWS CLI を介して AWS IoT Events API にアクセスできる必要があります。
- AWS IoT Core で、メッセージデータを AWS IoT Events に転送する AWS IoT Core ルールエンジンの [AWS IoT Events アクション](#) ルールを書き込みします。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 AWS IoT Core を介してメッセージを送信できる、またはすでに送信している場合は、この方法を使用します。この方法では、通常、デバイスからの処理能力が少なくても済みます。
- AWS IoT Analytics では、[CreateDataset](#) オペレーションを使用して、AWS IoT Events 入力を指定する `contentDeliveryRules` を使用してデータ・セットを作成します。このデータ・セットの内容は、データ・セットの内容が自動的に送信されます。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 (ルール) 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ตอบสนอง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として 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ための入力を作成します](#)
- [デバイスの状態を表すディテクターモデルを作成する](#)
- [ディテクターへの入力としてメッセージを送信する](#)

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

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) をキャプチャするための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。

デバイスの状態を表すディテクターモデルを作成する

[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](#) 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、適切な権限を持つ 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": "$variable.pressureThresholdBreachd > 1",
        "actions": [
          {
            "sns": {
              "targetArn": "arn:aws:sns:us-
east-1:123456789012:underPressureAction"
            }
          }
        ]
      }
    ]
  },
  "onInput": {
    "events": [
      {
        "eventName": "Overpressurized",
        "condition": "$input.PressureInput.sensorData.pressure > 70",
        "actions": [
          {
            "setVariable": {
              "variableName": "pressureThresholdBreachd",
              "value": "3"
            }
          }
        ]
      }
    ]
  },
  {
    "eventName": "Pressure Okay",
    "condition": "$input.PressureInput.sensorData.pressure <= 70",
    "actions": [
      {
        "setVariable": {
          "variableName": "pressureThresholdBreachd",
          "value": "$variable.pressureThresholdBreachd - 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"
}
```

ディテクターへの入力としてメッセージを送信する

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ための入力を作成します](#) を参照)。前のセクションでは、モーター

の過圧イベントに反応する detector model を作成しました ([デバイスの状態を表すディテクターモデルを作成する](#) 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。AWS IoT Events は、過去 5 分以内に送信された別のメッセージと同じ messageId を持っている場合、メッセージを無視します。

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":{
        "pressureThresholdBreach":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。

ディテクターモデルの制限と制限

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`
```

コメント例: HVAC 温度制御

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、[HVAC \(空気調和工学\) の温度制御](#)で入手できます。

背景

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。

入力定義

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。

ディテクターモデルの定義

'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:heat0ff"
                    }
                },
                {
                    "sns": {
                        "targetArn": "arn:aws:sns:us-west-2:123456789012:cool0ff"
                    }
                },
                {
                    "iotTopicPublish": {
                        "mqttTopic": "hvac/Heating/Off"
                    }
                },
                {
                    "iotTopicPublish": {
                        "mqttTopic": "hvac/Cooling/Off"
                    }
                }
            ]
        }
    ]
}
```



```
    "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:cool0n"
      }
    },
    {
      "iotTopicPublish": {
        "mqttTopic": "hvac/Cooling/0n"
      }
    },
    {
      "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:cool0ff"
      }
    },
    {
      "sns": {
        "targetArn": "arn:aws:sns:us-west-2:123456789012:heat0n"
      }
    },
    {
      "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:cool0ff"
                }
            },
            {
                "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 の例

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": []
}
```

BatchPutMessage の例

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 "areaId" を持つ AWS IoT Events が受信したメッセージにより、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": []
}
```

例: MQTT メッセージの取り込み

センサーコンピューティングリソースが "BatchPutMessage" API を使用できないが、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 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 "temp" フィールドから "sensorData.temperature" フィールドを取得し、このデータを AWS IoT Events "temperatureInput" に取り込む AWS IoT Events トピックルールの定義です。

センサーコンピューティングリソースが "BatchPutMessage" API を使用できないが、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 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 "temp" フィールドから "sensorData.temperature" フィールドを取得し、このデータを AWS IoT Events "temperatureInput" に取り込む AWS IoT Events トピックルールの定義です。

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/anotherRole"
        }
      }
    ],
    "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のようにデータが AWS IoT Events に取り込まれます。

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

ファイル: spooferExample.json

```
{
  "messages": [
    {
      "messageId": "54321",
      "inputName": "temperatureInput",

```

```

    "payload": "{\"sensorId\": \"03\", \"areaId\": \"Area51\", \"sensorData\":
  {\"temperature\": 24.5} }"
  }
]
}

```

例: 生成された Amazon SNS メッセージ

以下は、"Area51" デ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。

```

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

      "detectorModelName":"areaDetectorModel","keyValue":"Area51","detectorModelVersion":"1"}, "eventTime":1557520274729,
      {"inputName":"seedTemperatureInput","messageId":"00001","triggerType":"Message"},"state":{
        {"stateName":"start","variables":{
          {"sensorCount":10,"rangeHigh":30.0,"resetMe":false,"enteringNewState":true,"averageTemperature":24.5}}}}, "eventName":"resetHeatCool"}
    }
  }
}

```

```

Cooling system off command> {"eventTime":1557520274729,"payload":{
  {"actionExecutionId":"98f6a1b5-8f40-3cdb-9256-93afd4d66192","detector":{
    {"detectorModelName":"areaDetectorModel","keyValue":"Area51","detectorModelVersion":"1"}, "eventTime":1557520274729,
    {"inputName":"seedTemperatureInput","messageId":"00001","triggerType":"Message"},"state":{
      {"stateName":"start","variables":{
        {"sensorCount":10,"rangeHigh":30.0,"resetMe":false,"enteringNewState":true,"averageTemperature":24.5}}}}, "eventName":"resetHeatCool"}
  }
}

```

例: DescribeDetector API

DescribeDetector オペレーションを使用して、デテクターインスタンスの現在の状態、可変値、およびタイマーを確認できます。

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"
}
}
```

AWS IoT Core ルールエンジンの例

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。ディテクターモデルによって制御される各領域の加熱単位と冷却単位に対して AWS IoT Core が定義されていると想定します。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]

サポートされているアクション

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。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 リソースにデータを送信したりできます。

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、可変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 のサービス进行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。

組み込みアクションの使用

AWS IoT Events では、タイマーを使用したり、可変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、 は期間表現を再評価しません。

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。

他の AWS サービスの使用

AWS IoT Events では、AWS のサービス进行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

- と AWS サービスの両方 AWS IoT Events が連携するには、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

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、コンテンツ表現を使用します。詳細については、「AWS IoT Events API リファレンス」の「[表現](#)」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「[AWS IoT のメッセージブローカー](#)」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、「[AWS IoT Events のためのアイデンティティおよびアクセス管理](#)」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 [lotTopicPublishAction](#) 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のフリートのオペレーションの障害や変化をモニタリングし、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、以下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 `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、「[AWS IoT Events のためのアイデンティティおよびアクセス管理](#)」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 [lotEventsAction](#) 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、産業機器からデータを大規模に収集、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「[What is 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、以下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、ターゲットアセットプロパティを識別assetIdpropertyIdするために、propertyAliasまたはとの両方を指定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的識別子。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 `aws:iot: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、「[AWS IoT Events のためのアイデンティティおよびアクセス管理](#)」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 [lotSiteWiseAction](#) 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。デフォルトは STRING です。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。デフォルトは STRING です。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、「[AWS IoT Events のためのアイデンティティおよびアクセス管理](#)」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、「[AWS IoT Events のためのアイデンティティおよびアクセス管理](#)」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 などの宛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、以下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区切ることができます。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 `awsfirehose: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、「[AWS IoT Events のためのアイデンティティおよびアクセス管理](#)」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、サーバーをプロビジョニングまたは管理しなくてもコードを実行でき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 `aws: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、「[AWS IoT Events のためのアイデンティティおよびアクセス管理](#)」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、「[AWS IoT Events のためのアイデンティティおよびアクセス管理](#)」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 \(first in, first out\) 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 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) を指定した場合、 はデータを Base64 テキストに AWS IoT Events エンコードしますTRUE。デフォルトは FALSE です。

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 `aws: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、「[AWS IoT Events のためのアイデンティティおよびアクセス管理](#)」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 関数をトリガー AWS Lambda 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

大量のデータレコード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、初期分析を完了し、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。

表現

AWS IoT Events 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を提供します。表現を使用してリテラル値を指定するか、AWS IoT Events で特定の値を指定する前に式を評価できます。

構文

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。

リテラル

- 整数
- 10 進数
- 文字列
- 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): -

文字列

- 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)

- OR: |

例: $13 | 5 \rightarrow 13$

- AND: &

例: $13 \& 5 \rightarrow 5$

- XOR: ^

例: $13 \wedge 5 \rightarrow 8$

- NOT: ~

例: $\sim 13 \rightarrow -14$

ブール値

- 未満: <
- 以下: <=
- 等しい: ==
- 等しくない: !=
- 以上: >=
- 以上: >
- AND: &&
- OR: ||

 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。

関数

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。

例えば、ディテクターモデルには、TemperatureChecked と RecordUpdated の 2 つの状態があります。TemperatureChecked 状態でタイマーを作成しました。RecordUpdated 状態でタイマーを使用する前に、まず TemperatureChecked 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。

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は、参照の評価結果は String、Decimal、または Boolean である必要があります。

AWS IoT Events 表現は、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。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

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。expression は任意の AWS IoT Events 表現に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"
'{"sensor\":"\${\$variable.SensorID + 1}\}'	"{"sensor\":"11\"}"
'{"sensor\:">\${\$variable.SensorID + 1}}'	"{"sensor\:":11}"

表現の使用

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表現を入力します。
- 表現をパラメータとして AWS IoT Events API 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 表現を書き込み

AWS IoT Events 表現の書き込みに役立つ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ます。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
```

ディテクターモデルの例

このセクションには、ディテクターモデルおよび入力の例が含まれています。

トピック

- [HVAC \(空気調和工学\) の温度制御](#)
- [クレーン](#)
- [センサーとアプリケーションによるイベント検出](#)
- [デバイス HeartBeat](#)
- [ISA アラーム](#)
- [シンプルなアラーム](#)

HVAC (空気調和工学) の温度制御

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。(ディテクターインスタンスはエリアごとに作成されます。)
- 各ディテクターインスタンスは、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。

入力定義

"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。

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."
  }
}
```

ディテクターモデルの定義

"areaDetectorModel" は、ディテクターインスタ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:cool0n"
          }
        },
        {
          "iotTopicPublish": {
            "mqttTopic": "hvac/Cooling/0n"
          }
        },
        {
          "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 <=
$variable.anomalousLow",
      "actions": [
        {
          "iotTopicPublish": {
            "mqttTopic": "temperatureSensor/anomaly"
          }
        }
      ]
    },
    {
      "eventName": "highTemperatureSpike",
      "condition": "$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:cool0ff"
      }
    },
    {
      "sns": {
        "targetArn": "arn:aws:sns:us-west-2:123456789012:heat0n"
      }
    },
    {
      "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)) && $variable.goodToGo ==
true",
        "actions": [
            {
                "sns": {
                    "targetArn": "arn:aws:sns:us-west-2:123456789012:cool10ff"
                }
            },
            {
                "iotTopicPublish": {
                    "mqttTopic": "hvac/Cooling/Off"
                }
            }
        ],
        "nextState": "idle"
    }
]
}
},
{
    "stateName": "heating",
    "onEnter": {
        "events": [
            {
                "eventName": "delay",
                "condition": "!$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"
  }
}
```

BatchPutMessage の例

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"
}
}
```

BatchUpdateDetector の例

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。

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:

```
{
  "message": "An error occurred (InvalidRequestException) when calling the BatchUpdateDetector operation: Number of variables in the detector exceeds the limit 10"
}
```

AWS IoT Core ルールエンジンの例

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。ディテクターモデルによって制御される各領域の加熱単位と冷却単位に対して AWS IoT Core が定義されていると想定します。

この例では、"Area51HeatingUnit" および "Area51CoolingUnit" という名前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://ADMShadowCool10ffRule.json
```

ファイル: ADMShadowCool10ffRule.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]

クレーン

背景

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。

コマンド

```
#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
```

ディテクターモデル

ファイル: craneDetectorModel.json

```
{
  "detectorModelName": "craneDetectorModel",
  "detectorModelDefinition": {
    "states": [
      {
        "stateName": "Running",
        "onEnter": {
          "events": [
            {
```

```
        "eventName": "init",
        "condition": "true",
        "actions": [
            {
                "setVariable": {
                    "variableName": "craneThresholdBreach",
                    "value": "0"
                }
            }
        ]
    },
    ],
},
"onInput": {
    "events": [
        {
            "eventName": "Overheated",
            "condition": "$input.TemperatureInput.temperature > 35",
            "actions": [
                {
                    "setVariable": {
                        "variableName": "craneThresholdBreach",
                        "value": "$variable.craneThresholdBreach + 1"
                    }
                }
            ]
        },
        {
            "eventName": "Crane Threshold Breached",
            "condition": "$variable.craneThresholdBreach > 5",
            "actions": [
                {
                    "sns": {
                        "targetArn": "arn:aws:sns:us-
east-1:123456789012:CraneSNSTopic"
                    }
                }
            ]
        },
        {
            "eventName": "Underheated",
            "condition": "$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": "$input.TemperatureInput.temperature > 30",
        "actions": [
          {
            "setVariable": {
              "variableName": "craneThresholdBreach",
              "value": "$variable.craneThresholdBreach + 1"
            }
          }
        ]
      },
      {
        "eventName": "Crane Threshold Breached",
        "condition": "$variable.craneThresholdBreach > 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": "craneThresholdBreach",
```

```

        "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": "motorThresholdBreach",
                    "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": "motorThresholdBreach",
                        "value": "0"
                    }
                }
            ]
        }
    ],
    "onInput": {
        "events": [
            {
                "eventName": "Overheated And Overpressurized",
                "condition": "$input.PressureInput.pressure > 70 &&
$input.TemperatureInput.temperature > 30",
                "actions": [
                    {
                        "setVariable": {
                            "variableName": "motorThresholdBreach",
                            "value": "$variable.motorThresholdBreach + 1"
                        }
                    }
                ]
            },
            {
                "eventName": "Motor Threshold Breached",
                "condition": "$variable.motorThresholdBreach > 5",
                "actions": [
                    {
                        "sns": {
                            "targetArn": "arn:aws:sns:us-
east-1:123456789012:MotorSNSTopic"
                        }
                    }
                ]
            }
        ]
    }
}

```

```
    ],  
    "initialStateName": "Running"  
  },  
  "roleArn": "arn:aws:iam::123456789012:role/columboSNSRole"  
}
```

入力

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"}  
    ]  
  }  
}
```

メッセージ

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\"}"  
    }  
  ]  
}
```

```
    }  
  ]  
}
```

センサーとアプリケーションによるイベント検出

このディテクターモデルは、AWS IoT Events コンソールから利用できるテンプレートの 1 つです。便宜上、ここに再掲します。

```
{  
  "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": "Device_exception",  
              "actions": [  
                {  
                  "iotTopicPublish": {  
                    "mqttTopic": "Device_stolen"  
                  }  
                }  
              ],  
              "condition": "true"  
            }  
          ]  
        }  
      ]  
    }  
  }  
}
```

```
        "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"
}
}
```

デバイス HeartBeat

このディテクターモデルは、AWS IoT Events コンソールから利用できるテンプレートの1つです。便宜上、ここに再掲します。

```
{
  "detectorModelDefinition": {
    "states": [
      {
        "onInput": {
          "transitionEvents": [
            {
              "eventName": "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"
}
}
```

ISA アラーム

このディテクターモデルは、AWS IoT Events コンソールから利用できるテンプレートの 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"
}

```

シンプルなアラーム

このディテクターモデルは、AWS IoT Events コンソールから利用できるテンプレートの 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"
}
```

アラームによるモニタリング

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。アラーム状態が変化したときに発生する 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。AWS IoT Events は、データから派生した[入力属性](#)を適切なアラームに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。

AWS IoT SiteWise での作業

AWS IoT Events アラームを使用して、AWS IoT SiteWise 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 アラームに送信します。AWS IoT Events はアラーム状態を AWS IoT SiteWise に送信します。

AWS IoT SiteWise は外部アラームもサポートしています。AWS IoT SiteWise の外部でアラームを使用し、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で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。

アラームモデルの作成

AWS IoT Events アラームを使用してデータをモニタリングし、しきい値に違反し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、AWS IoT Events の入力属性または AWS IoT SiteWise のアセットプロパティをモニタリングできます。
- 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は、アラームモデルを作成する前に次のことを行ってください。
 - ステップ 1: 「[インプットの作成](#)」の概要を読む。
 - ステップ 2: 「[ナビゲーションペインの入力の作成](#)」手順を読む。
- アセットプロパティをモニタリングすることを選択した場合は、アラームモデルを作成する前に、AWS IoT SiteWise で[アセットモデルを作成する](#)必要があります。
- アラームがアクションを実行して 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 リージョンにある必要があります。

アラームモデルの作成 (コンソール)

次に、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。AWS IoT Events は、このキーに関連付けられた入力をアラームに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と同じ 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、および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、数値を入力するか、AWS IoT Events 入力で属性を選択します。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、「[受信者の管理](#)」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が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 が自動的に IAM ロールを作成します。
- 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. 追加の通知設定については、AWS Lambda 関数を編集してアラーム通知を管理できます。AWS Lambda 関数の次のオプションのいずれかを選択します。
- 新しい AWS Lambda 関数を作成します - 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 AWS Lambda 関数を使用します。

可能なアクションの詳細については、「[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の場合、アラーム状態が変化したときに実行する 1 つ以上の AWS IoT Events アクションを追加できます。
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. [作成] を選択します。

アラームへの対応

[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。

アラームへの対応 (コンソール)

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。

アラームへの対応 (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](#)

アラーム通知の管理

AWS IoT Events は Lambda 関数を使用してアラーム通知を管理します。AWS IoT Events が提供する Lambda 関数を使用するか、新しい関数を作成できます。

Lambda 関数の作成

AWS IoT Events は、アラームが電子メールおよび 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 E メールまたは SMS 通知を送信する場合は、AWS Lambda が Amazon SES および Amazon SNS と連携できるようにする 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"
      ],
      "Resource": "*"
    },
    {
      "Effect": "Deny",
      "Action": [
        "sns:Publish"
      ],
      "Resource": "arn:aws:sns:*:*:*"
    }
  ]
}
```

- AWS IoT Events と AWS Lambda の両方に同じ 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](#)」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。

Lambda 関数のデプロイ

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。

以下に、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ます。

1. デバイスの端末で `aws --version` を実行して、AWS CLI がインストールされ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「[AWS CLI のインストール](#)」を参照してください。
2. `aws configure list` を実行して、このチュートリアルのすべての AWS リソースを持つ AWS リージョンで AWS CLI を設定した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「[AWS CLI の設定](#)」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つ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。

カスタム Lambda 関数の作成

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て AWS リソースにアクセスできるようにする許可を追加します。
- AWS IoT Events が提供する Lambda 関数を使用する場合は、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 SS0. 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.utcnow().timestamp(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"
              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 SS0.
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.utcnow().timestamp(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

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「[Amazon SES での E メールアドレスの確認](#)」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。

AWS IoT Events が提供する Lambda 関数は、E.164 形式の電話番号を使用しているかどうかをチェックします。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。

受信者の管理

AWS IoT Events は、AWS IAM Identity Center (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。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 に同じ AWS リージョンを選択する必要があります。
- AWS 組織は、一度に 1 つの IAM Identity Center 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。

AWS IoT Events のセキュリティ

AWS では、クラウドのセキュリティが最優先事項です。AWS の顧客は、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利点を得られます。

セキュリティは、AWS と顧客の間の責任共有です。[責任共有モデル](#)では、これをクラウドのセキュリティおよびクラウド内のセキュリティとして説明しています。

- クラウドのセキュリティ - AWS は、AWS 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責任を負います。また、AWS は、使用するサービスを安全に提供します。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。AWS IoT Events に適用されるコンプライアンスプログラムの詳細については、[コンプライアンスプログラム対象範囲内の AWS 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 の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、AWS IoT Events を使用するとき、責任共有モデルを適用する方法を理解するのに役立ちます。次のトピックでは、セキュリティおよびコンプライアンスの目的を達成するために AWS IoT Events を設定する方法を示します。また、AWS IoT Events リソースのモニタリングと保護に役立つ他の AWS サービスの使用方法についても説明します。

トピック

- [AWS IoT Events のためのアイデンティティおよびアクセス管理](#)
- [AWS IoT Events のモニタリング](#)
- [のコンプライアンス検証 AWS IoT Events](#)
- [AWS IoT Events での耐障害性](#)
- [AWS IoT Events でのインフラストラクチャセキュリティ](#)

AWS IoT Events のためのアイデンティティおよびアクセス管理

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コントロールするために役立つ AWS のサービスです。IAM 管理者は、誰を認証 サインインし、誰

に AWS IoT Events リソースの使用を許可する アクセス許可を持たせるかを制御します。IAM では、AWS のサービスで追加料金は発生しません。

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権の管理](#)
- [詳細はこちら](#)
- [AWS IoT Events と IAM の連携方法](#)
- [AWS IoT Events アイデンティティベースポリシーの例](#)
- [サービス間の混乱した代理の防止](#)
- [AWS IoT Events Identity and Access のトラブルシューティング](#)

対象者

AWS Identity and Access Management (IAM) の用途は、AWS IoT Events で行う作業によって異なります。

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は、必要な権限と認証情報を管理者が用意します。作業を実行するためにさらに多くの AWS IoT Events 機能を使用するとき、追加の権限が必要になる場合があります。アクセスの管理方法を理解すると、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 Identity and Access 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、AWS IoT Events への完全なアクセスがあります。サービスのユーザーがどの 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確認して、IAM の基本概念を理解してください。お客様の会社で AWS IoT Events で IAM を利用する方法の詳細については、「[AWS IoT Events と IAM の連携方法](#)」を参照してください。

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、「[AWS IoT Events アイデンティティベースポリシーの例](#)」を参照してください。

アイデンティティを使用した認証

認証とは、アイデンティティ認証情報を使用して AWS にサインインする方法です。ユーザーは、AWS アカウントのルートユーザーもしくは IAM ユーザーとして、または IAM ロールを引き受けることによって、認証を受ける (AWS にサインインする) 必要があります。

ID ソースから提供された認証情報を使用して、フェデレーテッドアイデンティティとして AWS にサインインできます。AWS IAM Identity Center フェデレーテッドアイデンティティの例としては、(IAM Identity Center) ユーザー、会社のシングルサインオン認証、Google または Facebook の認証情報などがあります。フェデレーテッドアイデンティティとしてサインインする場合、IAM ロールを使用して、前もって管理者により ID フェデレーションが設定されています。フェデレーションを使用して AWS にアクセスする場合、間接的にロールを引き受けることになります。

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。AWS へのサインインの詳細については、『AWS サインイン ユーザーガイド』の「[AWS アカウントにサインインする方法](#)」を参照してください。

プログラムで AWS にアクセスする場合、AWS は Software Development Kit (SDK) とコマンドラインインターフェイス (CLI) を提供し、認証情報で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「[AWS API リクエストの署名](#)」を参照してください。

使用する認証方法を問わず、追加のセキュリティ情報の提供が求められる場合もあります。例えば、AWS では、アカウントのセキュリティ強化のために多要素認証 (MFA) の使用をお勧めしています。詳細については、「AWS IAM Identity Center ユーザーガイド」の「[Multi-factor authentication \(多要素認証\)](#)」および「IAM ユーザーガイド」の「[AWS での多要素認証 \(MFA\) の使用](#)」を参照してください。

AWS アカウントのルートユーザー

AWS アカウントを作成する場合は、そのアカウントのすべての AWS のサービスとリソースに対して完全なアクセス権を持つ 1 つのサインインアイデンティティから始めます。このアイデンティティは AWS アカウントの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によってアクセスできます。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1つのアプリケーションに対して特定の権限を持つ AWS アカウント内のアイデンティティです。可能であれば、パスワードやアクセスキーなどの長期的な認証情報を保有する IAM ユーザーを作成する代わりに、一時的な認証情報を使用することをお勧めします。ただし、IAM ユーザーでの長期的な認証情報が必要な特定のユースケースがある場合は、アクセスキーをローテーションすることをお勧めします。詳細については、「IAM ユーザーガイド」の「[長期的な認証情報を必要とするユースケースのためにアクセスキーを定期的にローテーションする](#)」を参照してください。

[IAM グループ](#)は、IAM ユーザーの集団を指定するアイデンティティです。グループとしてサインインすることはできません。グループを使用して、複数のユーザーに対して一度に権限を指定できます。多数のユーザーグループがある場合、グループを使用することで権限の管理が容易になります。例えば、IAMAdmins という名前のグループを設定して、そのグループに IAM リソースを管理する権限を与えることができます。

ユーザーは、ロールとは異なります。ユーザーは1人の人または1つのアプリケーションに一意に関連付けられますが、ロールはそれを必要とする任意の人が引き受けるようになっています。ユーザーには永続的な長期の認証情報がありますが、ロールでは一時的な認証情報が提供されます。詳細については、「IAM ユーザーガイド」の「[IAM ユーザー \(ロールではなく\) の作成が適している場合](#)」を参照してください。

IAM ロール

[IAM ロール](#)は、特定の権限を持つ、AWS アカウント内のアイデンティティです。これは IAM ユーザーに似ていますが、特定のユーザーには関連付けられていません。[ロールを切り替える](#)ことによって、AWS Management Console で IAM ロールを一時的に引き受けることができます。ロールを引き受けるには、AWS CLI または AWS API オペレーションを呼び出すか、カスタム URL を使用します。ロールを使用する方法の詳細については、「IAM ユーザーガイド」の「[IAM ロールの使用](#)」を参照してください。

一時的な認証情報を持った IAM ロールは、以下の状況で役立ちます。

- フェデレーションユーザーユーザーアクセス – フェデレーションアイデンティティに権限を割り当てるには、ロールを作成してそのロールの権限を定義します。フェデレーションアイデンティティが認証されると、そのアイデンティティはロールに関連付けられ、ロールで定義されている権限が付与されます。フェデレーションの詳細については、「IAM ユーザーガイド」の「[サードパーティ ID プロバイダー向けロールの作成](#)」を参照してください。IAM アイデンティティセンターを使用する場合、権限セットを設定します。アイデンティティが認証後にアクセスできるもの

を制御するため、IAM Identity Center は、権限セットを IAM のロールに関連付けます。権限セットの詳細については、「AWS IAM Identity Center ユーザーガイド」の「[権限セット](#)」を参照してください。

- 一時的な IAM ユーザー権限 - IAM ユーザーまたはロールは、特定のタスクに対して複数の異なる権限を一時的に IAM ロールで引き受けることができます。
- クロスアカウントアクセス - IAM ロールを使用して、自分のアカウントのリソースにアクセスすることを、別のアカウントの人物 (信頼済みプリンシパル) に許可できます。クロスアカウントアクセス権を付与する主な方法は、ロールを使用することです。ただし、一部の AWS のサービスでは、(ロールをプロキシとして使用する代わりに) リソースにポリシーを直接アタッチできます。クロスアカウントアクセスにおけるロールとリソースベースのポリシーの違いについては、「IAM ユーザーガイド」の「[IAM ロールとリソースベースのポリシーとの相違点](#)」を参照してください。
- クロスサービスアクセス - 一部の AWS のサービスでは、他の AWS のサービスの機能を使用します。例えば、あるサービスで呼び出しを行うと、通常そのサービスによって Amazon EC2 でアプリケーションが実行されたり、Amazon S3 にオブジェクトが保存されたりします。サービスでは、呼び出し元プリンシパルの権限、サービスロール、またはサービス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 AWS でアクションを実行するユーザーは、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、AWS のサービスを呼び出すプリンシパルの権限を、AWS のサービスのリクエストと合わせて使用し、ダウンストリームのサービスに対してリクエストを行います。FAS リクエストは、サービスが、完了するために他の AWS のサービス または リソースとのやりとりを必要とするリクエストを受け取ったときにのみ行われます。この場合、両方のアクションを実行するためのアクセス許可が必要です。FAS リクエストを行う際のポリシーの詳細については、「[転送アクセスセッション](#)」を参照してください。
- サービスロール - サービスがユーザーに代わってアクションを実行するために引き受ける [IAM ロール](#)です。IAM 管理者は、IAM 内からサービスロールを作成、変更、削除できます。詳細については、「IAM ユーザーガイド」の「[AWS のサービスにアクセス許可を委任するロールの作成](#)」を参照してください。
- サービスリンクロール - サービスリンクロールは、AWS のサービスにリンクされたサービスロールの一種です。サービスがロールを引き受け、ユーザーに代わってアクションを実行できるようになります。サービスリンクロールは、AWS アカウントに表示され、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権限を表示できますが、編集することはできません。

- Amazon EC2 で実行されているアプリケーション - EC2 インスタンスで実行され、AWS CLI または AWS API 要求を行っているアプリケーションの一時的な認証情報を管理するには、IAM ロールを使用できます。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、インスタンスに添付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 ロールを使用してアクセス許可を付与する](#)」を参照してください。

IAM ロールと IAM ユーザーのどちらを使用するかについては、「IAM ユーザーガイド」の「[\(IAM ユーザーではなく\) IAM ロールをいつ作成したら良いのか?](#)」を参照してください。

ポリシーを使用したアクセス権の管理

AWS でアクセス権を管理するには、ポリシーを作成して AWS アイデンティティまたはリソースにアタッチします。ポリシーは AWS のオブジェクトであり、アイデンティティやリソースに関連付けて、これらの権限を定義します。AWS は、プリンシパル (ユーザー、ルートユーザー、またはロールセッション) がリクエストを行うと、これらのポリシーを評価します。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。大半のポリシーは JSON ドキュメントとして AWS 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 AWSJSON ポリシーを使用して、だれが何にアクセスできるかを指定できます。つまり、どのプリンシパルがどのリソースに対してどのような条件下でアクションを実行できるかということです。

デフォルトでは、ユーザーやロールに権限はありません。IAM 管理者は、リソースで必要なアクションを実行するための権限をユーザーに付与する IAM ポリシーを作成できます。その後、管理者はロールに IAM ポリシーを追加し、ユーザーはロールを引き継ぐことができます。

IAM ポリシーは、オペレーションの実行方法を問わず、アクションの権限を定義します。例えば、iam:GetRole アクションを許可するポリシーがあるとします。このポリシーがあるユーザーは、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ポリシー

アイデンティティベースポリシーは、IAM ユーザー、ユーザーのグループ、ロールなど、アイデンティティにアタッチできる JSON 権限ポリシードキュメントです。これらのポリシーは、ユーザー

とロールが実行できるアクション、リソース、および条件を制御します。アイデンティティベースのポリシーを作成する方法については、「IAM ユーザーガイド」の「[IAM ポリシーの作成](#)」を参照してください。

アイデンティティベースポリシーは、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ます。管理ポリシーは、AWS アカウント内の複数のユーザー、グループ、およびロールにアタッチできるスタンドアロンポリシーです。マネージドポリシーには、AWS マネージドポリシーとカスタマー管理ポリシーがあります。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「[マネージドポリシーとインラインポリシーの比較](#)」を参照してください。

他のポリシータイプ

AWS では、他の一般的ではない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。

- **権限の境界** - 権限の境界は、アイデンティティベースのポリシーによって IAM エンティティ (IAM ユーザーまたはロール) に付与できる許可の上限を設定する高度な機能です。エンティティに権限の境界を設定できます。結果として得られる権限は、エンティティのアイデンティティベースポリシーとその権限の境界の共通部分になります。Principal フィールドでユーザーまたはロールを指定するリソースベースのポリシーでは、権限の境界は制限されません。これらのポリシーのいずれかを明示的に拒否した場合、権限は無効になります。権限の境界の詳細については、「IAM ユーザーガイド」の「[IAM エンティティの権限の境界](#)」を参照してください。
- **サービスコントロールポリシー (SCP)** - SCP は、AWS Organizations で組織や組織単位 (OU) の最大権限を指定する JSON ポリシーです。AWS Organizations は、顧客の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、一元的に管理するサービスです。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 はメンバーアカウントのエンティティに対する権限を制限します (各 AWS アカウントのルートユーザーなど)。Organizations と SCP の詳細については、『AWS Organizations ユーザーガイド』の「[SCP の仕組み](#)」を参照してください。
- **セッションポリシー** - セッションポリシーは、ロールまたはフェデレーションユーザーの一時的なセッションをプログラムで作成する際にパラメータとして渡す高度なポリシーです。結果としてセッションの権限の範囲は、ユーザーまたはロールのアイデンティティベースポリシーとセッションポリシーの共通部分になります。また、リソースベースのポリシーから権限が派生する場合があります。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](#)」をご参照ください。

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している場合に AWS がリクエストを許可するかどうか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。

詳細はこちら

AWS IoT Events の Identity and Access Management に関する詳細については、以下のページに進んでください。

- [AWS IoT Events と IAM の連携方法](#)
- [AWS IoT Events Identity and Access のトラブルシューティング](#)

AWS IoT Events と IAM の連携方法

AWS IoT Events へのアクセスを管理するために IAM を使用する前に、AWS IoT Events での IAM 特徴が使用できるかを理解しておく必要があります。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「[IAM と連携する AWS のサービス](#)」をご参照ください。

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lement 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。

AWS IoT Events のポリシーアクションは、アクションの前に以下のプレフィックスを使用します。iotevents:。例えば、AWS IoT Events CreateInput API オペレーションを使用して AWS IoT Events 入力を作成する許可を誰かに付与するには、そのユーザーのポリシーに iotevents:CreateInput アクションを含めます。AWS IoT Events BatchPutMessage API オペレーションで入力を送信する許可を誰かに付与するには、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 要素または NotAction 要素のいずれかを含める必要があります。AWS IoT Events は、このサービスで実行できるタスクを説明する独自の一連のアクションを定義します。

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、次のようにカンマで区切ります。

```
"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表示するには、[IAM ユーザーガイド](#)のAWS IoT Events によって定義されたアクションを参照してください。

リソース

Resource エlement 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lement 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 デテクターモデルリソースの ARN は以下のようになります。

```
arn:${Partition}:iotevents:${Region}:${Account}:detectorModel/${detectorModelName}
```

ARN の形式の詳細については、「[Amazon リソースネーム \(ARN\) と AWS のサービスの名前空間](#)」を参照してください。

例えば、ステートメントで Foobar デ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、特定のリソースでは実行できません。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、複数のリソースが関連します。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テクターモデルを使用する権限が必要です。複数のリソースを単一の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 ARN のリストを表示するには、IAM ユーザーガイドの [AWS IoT Events で定義されるリソース](#) を参照してください。どのアクションで各リソースの ARN を指定できるかについては、[AWS 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lement 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lement 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lement を指定する場合、または 1つの Condition エlement 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进行评估します。単一の条件キーに複数の値を指定すると、AWS は OR 論理演算子を使用して条件进行评估します。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。たとえば、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 にはサービス固有条件キーがありませんが、いくつかのグローバル条件キーの使用がサポートされています。すべての AWS グローバル条件キーの一覧は、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、[AWS IoT Events アイデンティティベースポリシーの例](#)を参照してください。

AWS IoT Events リソースベースのポリシー

AWS IoT Events では、リソースベースのポリシーはサポートされていません。」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

タグを AWS IoT Events リソースにアタッチすることも、AWS IoT Events へのリクエストでタグを渡すこともできます。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、「[タグに基づくAWS IoT Events##の表示](#)」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を持つ、AWS アカウント 内のエンティティです。

AWS IoT Events での一時的な認証情報の使用

テンポラリ認証情報を使用して、フェデレーションでサインイン、IAM ロールを引き受ける、またはクロスアカウントロール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](#)やなどの AWS Security Token Service (AWS STS) API オペレーションを呼び出します。[GetFederationToken](#)。

AWS IoT Events はテンポラリ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ロール

[サービスリンクロール](#)は、AWS サービスが他のサービスのリソースにアクセスしてお客様の代わりにアクションを完了することを許可します。サービスにリンクされたロールは 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、サービスにリンクされた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 で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。

サービスロール

この機能により、ユーザーに代わってサービスが[サービスロール](#)を引き受けることが許可されます。このロールにより、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。サービスロールは、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、このロール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ポリシーの例

デフォルトでは、ユーザーおよびロールには、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。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もできません。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

- [ポリシーのベストプラクティス](#)
- [AWS IoT Events コンソールを使用する](#)
- [自分の許可の表示をユーザーに許可する](#)
- [AWS IoT Events 入力にアクセスする](#)
- [タグに基づくAWS IoT Events入力の表示](#)

ポリシーのベストプラクティス

アイデンティティベースのポリシーは非常に強力です。アカウント内で、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 アカウントに追加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 管理ポリシーの使用をスタートする - AWS IoT Events の使用をすばやくスタートするには、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すで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を使用した許可の使用スタート](#)」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いリソースまたは 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、AWS アカウントの AWS IoT Events リソースの詳細をリストおよび表示できます。最小限の必要なアクセス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なくなります。

これらのエンティティが AWS IoT Events コンソールを使用できるように、次の AWS 管理ポリシーもエンティティにアタッチします。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。

自分の許可の表示をユーザーに許可する

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。このポリシーには、コンソールで、またはプログラムで AWS CLI または AWS API を使用してこのアクションを完了するための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:ListGroupForUser",
        "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": "*"
}
]
```

AWS IoT Events 入力にアクセスする

この例では、AWS アカウント のユーザーに、AWS IoT Events 入力の 1 つである `exampleInput` へのアクセス権限を付与したいとします。また、ユーザーが入力を追加、更新、および削除できるようにしたいとします。

このポリシー

は、`iotevents:ListInputs`、`iotevents:DescribeInput`、`iotevents:CreateInput`、`iotevents:DeleteInput` の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権限を付与し、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) のチュートリアル例については、「[チュートリアル例: ユーザーポリシーを使用してバケットへのアクセスを制御する](#)」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/*"
}
]
}
```

タグに基づくAWS IoT Events##の表示

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コントロールできます。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。richard-roe という名前のユーザーが AWS IoT Events `##` を表示しようとする場合、`##` には `Owner=richard-roe` または `owner=richard-roe` のタグを付ける必要があります。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。

サービス間の混乱した代理の防止

Note

- AWS IoT Events サービスは、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ロールの使用しか許可しません。これは、このサービスでは混乱した副攻撃を実行できないことを意味します。
- 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顧客が確認するためのリファレンスとして機能し、AWS IoT Events サービスでクロスアカウントリソースが許可された場合に防ぐことができます。

混乱した副問題は、アクションを実行する許可を持たないエンティティが、より特権のあるエンティティにアクションを実行するように強制できるセキュリティの問題です。AWS では、サービス間でのなりすましが、混乱した代理問題を生じさせることがあります。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、別の顧客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、AWS では、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ですべてのサービスのデータを保護するために役立つツールを提供しています。

AWS IoT Events がリソースに別のサービスを提供する権限を制限するために、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することをお勧めします。aws:SourceArn 値にアカウント ID (Amazon S3 バケット ARN など) 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には、グローバルコンテキスト条件キー `aws:SourceArn` で、ARN の未知部分を示すためにワイルドカード (*) を使用します。例えば、`arn:aws:iotevents:*:123456789012:*` です。

次の例は、AWS IoT Events 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、混乱した代理問題を防ぐ方法を示しています。

トピック

- [例 1: デテクターモデルへのアクセス](#)
- [例 2: アラームモデルへのアクセス](#)
- [例 3: 指定したリージョン内のリソースへのアクセス](#)
- [例 4: ログオプション](#)

例 1: デテクターモデルへのアクセス

次のロールは、`foo` という名前の `DetectorModel` へのアクセスにのみ使用できます。

```
{
  "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/foo"
      }
    }
  }
]
}
```

例 2: アラームモデルへのアクセス

次のロールは、任意のアラームモデルへのアクセスにのみ使用できます。

```
{
  "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:alarmModel/*"
        }
      }
    }
  ]
}
```

例 3: 指定したリージョン内のリソースへのアクセス

次の例は、指定したリージョンのリソースにアクセスするために使用できるロールを示しています。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 4: ログオプション

ログ記録オプションのロールは、IoT Events のすべてのリソースに対して許可する必要があります。したがって、リソースタイプとリソース名にワイルドカード (*) を使用する必要があります。

```
{  
  "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 Identity and Access のトラブルシューティング

以下の情報は、AWS IoT Events と IAM の使用に伴って発生する可能性がある一般的な問題の診断や修復に役立ちます。

トピック

- [AWS IoT Events でアクションを実行する権限がない](#)
- [iam:PassRole を実行する権限がない](#)
- [自分の AWS アカウント 以外のユーザーに AWS IoT Events リソースへのアクセスを許可したい](#)

AWS IoT Events でアクションを実行する権限がない

AWS Management Console から、アクションを実行する認可がないと通知された場合、管理者に問い合わせ、サポートを依頼する必要があります。管理者とは、ユーザーにユーザー名とパスワードを提供した人です。

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やサービスリンクロールを作成せずに、既存のロールをサービスに渡すことができます。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と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。Mary には、ロールをサービスに渡す権限がありません。

```
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、Mary に iam:PassRole アクションの実行を許可する必要があります。

サポートが必要な場合は、AWS 管理者に問い合わせてください。サインイン認証情報を提供した担当者が管理者です。

自分の AWS アカウント 以外のユーザーに AWS IoT Events リソースへのアクセスを許可したい

他のアカウントのユーザーや組織外のユーザーが、リソースへのアクセス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制御リスト (ACL) をサポートするサービスの場合、それらのポリシーを使用して、リソースへのアクセスを付与できます。

詳細については、以下を参照してください。

- AWS IoT Events がこれらの機能をサポートしているかどうかを確認するには、「[AWS IoT Events と IAM の連携方法](#)」をご参照ください。
- 所有している AWS アカウント 全体のリソースへのアクセス権を付与する方法については、「IAM ユーザーガイド」の「[所有している別の AWS アカウント アカウントへのアクセス権を IAM ユーザーに付与する](#)」を参照してください。
- サードパーティーの AWS アカウント にリソースへのアクセス権を提供する方法については、『IAM ユーザーガイド』の「[第三者が所有する AWS アカウント へのアクセス権を付与する](#)」を参照してください。
- ID フェデレーションを介してアクセスを提供する方法については、『IAM ユーザーガイド』の「[外部で認証されたユーザー \(ID フェデレーション\) へのアクセスの許可](#)」を参照してください。
- クロスアカウントアクセスでのロールとリソースベースのポリシーの使用の違いの詳細については、『IAM ユーザーガイド』の「[IAM ロールとリソースベースのポリシーとの相違点](#)」を参照してください。

AWS IoT Events のモニタリング

モニタリングは、AWS IoT Events と 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パートです。マルチポイント障害が発生した場合に、より容易にデバッグできるよう、AWS ソリューションのすべてのパートからモニタリングのデータを収集することをお勧めします。ただし、AWS IoT Events のモニタリングをスタートする前に、以下の質問に対する回答を反映した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ロード条件でパフォーマンスを測定することにより、お客様の環境で通常の AWS IoT Events のパフォーマンスのベースラインを確定します。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

- [モニタリングツール](#)
- [Amazon CloudWatch でのモニタリング](#)
- [AWS CloudTrail を使用した AWS IoT Events API呼び出しのロギング](#)

モニタリングツール

AWS は、AWS IoT Events のモニタリングに使用できるさまざまなツールを提供します。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

以下の自動化されたモニタリングツールを使用して、AWS IoT Events をモニタリングし、問題が発生したときにレポートできます。

- Amazon CloudWatch Logs - AWS CloudTrail またはその他の出典のログファイルのモニタリング、保存、アクセスを行います。詳細については、Amazon CloudWatch ユーザーガイドの[ログファイルのモニタリング](#)を参照してください。
- Amazon CloudWatch Events - イベントに一致したものを 1 つ以上のターゲットの関数またはストリームに渡して、変更、状態の情報の収集、是正措置を行います。詳細については、Amazon CloudWatch ユーザーガイドの[Amazon CloudWatch Events とは](#)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、CloudTrail ログファイルを CloudWatch Logs に送信して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作成し、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検証します。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

AWS IoT Events のモニタリングでもう 1 つ重要なパートは、CloudWatch のアラームの対象外の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は、AWS 環境の状態を一目で確認できます。AWS IoT Events のログファイルを確認することもお勧めします。

- 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 を使用して以下のことを行えます。

- 重視するサービスをモニタリングするための[カスタマイズしたダッシュボード](#)を作成します
- 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
- AWS リソースのすべての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 でのモニタリング

AWS IoT Events デテクターモデルの開発やデバッグをする時、AWS IoT Events が何を行い、どのようなエラーに遭遇したかを知る必要があります。Amazon CloudWatch は、Amazon Web Services (AWS) リソースと、AWS で実行され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、運用ヘルスをシステム全体で把握できます。[AWS IoT Events デテクターモデルの開発時に Amazon CloudWatch ログを有効にする](#)には、AWS IoT Events の CloudWatch ロ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
    }
  ]
}
```

AWS CloudTrail を使用した AWS IoT Events API 呼び出しのロギング

AWS IoT Events は AWS CloudTrail と統合されています。このサービスは、ユーザーやロール、または AWS の AWS IoT Events のサービスによって実行されたアクションを記録するサービスです。CloudTrail は、AWS IoT Events コンソールからの呼び出しや AWS IoT Events API へのコード呼び出しを含む、AWS IoT Events のすべての API コールをイベントとしてキャプチャします。

トレイルを作成すると、AWS IoT Events のイベントを含む CloudTrail イベントの Amazon S3 ケットへの継続的な配信が可能になります。証跡を設定しない場合でも、CloudTrail コンソールの [Event history (イベント履歴)] で最新のイベントを表示できます。CloudTrail によって収集された情報を使用して、AWS IoT Events に対して行われた要求、要求が行われた IP アドレス、要求を行った人、要求が行われた日時、および追加の詳細を判別できます。

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。

CloudTrail の AWS IoT Events 情報

CloudTrail は、アカウントを作成すると AWS アカウントで有効になります。AWS IoT Events でアクティビティが発生すると、そのアクティビティは CloudTrail イベントにレコードされ、他の AWS サービスイベントがイベント履歴にレコードされます。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 イベント履歴でのイベントの表示](#)」をご参照ください。

AWS のイベントなど、AWS IoT Events アカウントのイベントの継続的なレコードについては、追跡を作成します。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、コンソールで追跡を作成するときに、追跡がすべての AWS リージョンに適用されます。追跡は 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 ログで収集したデータをより詳細に分析し、それに基づく対応を行うよう、他の AWS サービスを設定することができます。詳細については、以下をご覧ください。

- [追跡を作成するための概要](#)
- [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、リクエストの生成者に関する情報が含まれます。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 のサービスによって送信されたかどうか。

詳細については、「[CloudTrail userIdentity 要素](#)」を参照してください。AWS IoT Events アクションは、[AWS IoT Events API リファレンス](#)に記載されています。

AWS IoT Events ログファイルエントリについて

追跡は、指定した Amazon S3 バケットにログファイルとしてイベントを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ています。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、他の AWS サービスレコードとともに書き込まれる CloudTrail ログファイルで追跡されます。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 のサービスによって送信されたかどうか。

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。

また、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[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。

次の例は、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"
}
```

次の例は、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",
  "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"
}
```

次の例は、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",
  "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"
}
```

次の例は、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"
}
```

次の例は、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"
}
```

次の例は、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"
}
```

次の例は、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"
}
```

次の例は、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"
}
```

次の例は、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": "CkZEZXR1Y3Rvck1vZGVsM19saXN0ZGV0ZWNo0b3Jtb2R1bHN0ZXN0X2V10WJkZTk1YT",
  "maxResults": 3
},
"responseElements": null,
"requestID": "6d70f262-da95-4bb5-94b4-c08369df75bb",
"eventID": "2d01a25c-d5c7-4233-99fe-ce1b8ec05516",
"eventType": "AwsApiCall",
"recipientAccountId": "123456789012"
}
```

次の例は、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": "ebeb277-6bd8-44ea-8abd-fbf40ac044ee",
"eventID": "fc6281a2-3fac-4e1e-98e0-ca6560b8b8be",
"eventType": "AwsApiCall",
"recipientAccountId": "123456789012"
}
```

次の例は、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"
}
```

次の例は、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": "CkhjYW5hcnlfdGVzdF9pbnB1dF9saXN0ZGV0ZWNo0b3Jtb2R1bHN0ZXN0ZDU3OGZ",
  "maxResults": 3
},
"responseElements": null,
"requestID": "dd6762a1-1f24-4e63-a986-5ea3938a03da",
"eventID": "c500f6d8-e271-4366-8f20-da4413752469",
"eventType": "AwsApiCall",
"recipientAccountId": "123456789012"
}
```

次の例は、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"
}
```

次の例は、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"
}
```

次の例は、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"
}
```

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、コンプライアンスプログラム [AWS のサービス による対象範囲内のコンプライアンスプログラム](#) を参照し、関心のあるコンプライアンスプログラムを選択します。一般的な情報については、[AWS 「コンプライアンスプログラム」](#) を参照してください。

を使用して、サードパーティーの監査レポートをダウンロードできます AWS Artifact。詳細については、「[でのレポートのダウンロード AWS Artifact](#)」の「」を参照してください。

を使用する際のお客様のコンプライアンス責任 AWS のサービス は、お客様のデータの機密性、貴社のコンプライアンス目的、適用される法律および規制によって決まります。は、コンプライアンスに役立つ以下のリソース AWS を提供しています。

- [セキュリティとコンプライアンスのクイックスタートガイド](#) – これらのデプロイガイドでは、アーキテクチャ上の考慮事項について説明し、セキュリティとコンプライアンスに重点を置いたベースライン環境 AWS を にデプロイする手順について説明します。
- [アマゾン ウェブ サービスにおける HIPAA セキュリティとコンプライアンスのアーキテクチャー](#) – このホワイトペーパーでは、企業が AWS を使用して HIPAA 対象アプリケーションを作成する方法について説明します。

Note

すべて AWS のサービス HIPAA の対象となるわけではありません。詳細については、「[HIPAA 対応サービスのリファレンス](#)」を参照してください。

- [AWS コンプライアンスリソース](#) – このワークブックとガイドのコレクションは、お客様の業界や地域に適用される場合があります。
- [AWS カスタマーコンプライアンスガイド](#) – コンプライアンスの観点から責任共有モデルを理解します。このガイドでは、ガイダンスを保護し AWS のサービス、複数のフレームワーク (米国立標準技術研究所 (NIST)、Payment Card Industry Security Standards Council (PCI)、国際標準化機構 (ISO) を含む) のセキュリティコントロールにマッピングするためのベストプラクティスをまとめています。
- 「[デベロッパーガイド](#)」の「[ルールによるリソースの評価](#)」 – この AWS Config サービスは、リソース設定が社内プラクティス、業界ガイドライン、および規制にどの程度準拠しているかを評価します。AWS Config

- [AWS Security Hub](#) – これにより AWS のサービス、内のセキュリティ状態を包括的に確認できます AWS。Security Hub では、セキュリティコントロールを使用して AWS リソースを評価し、セキュリティ業界標準とベストプラクティスに対するコンプライアンスをチェックします。サポートされているサービスとコントロールのリストについては、「[Security Hub のコントロールリファレンス](#)」を参照してください。
- [Amazon GuardDuty](#) – これにより AWS アカウント、疑わしいアクティビティや悪意のあるアクティビティがないか環境を監視することで、ワークロード、コンテナ、データに対する潜在的な脅威 AWS のサービスを検出します。GuardDuty は、特定のコンプライアンスフレームワークで義務付けられている侵入検知要件を満たすことで、PCI DSS などのさまざまなコンプライアンス要件への対応に役立ちます。
- [AWS Audit Manager](#) – これにより AWS のサービス、AWS 使用状況を継続的に監査し、リスクの管理方法と規制や業界標準への準拠を簡素化できます。

AWS IoT Events での耐障害性

AWS の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ゾーンを中心に構築されます。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ゾーンがあります。アベイラビリティゾーンでは、アベイラビリティ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、および拡張性が優れています。

AWS リージョンとアベイラビリティゾーンの詳細については、「[AWS グローバルインフラストラクチャ](#)」を参照してください。

AWS IoT Events でのインフラストラクチャセキュリティ

マネージドサービスである AWS 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 AWS がインフラストラクチャを保護する方法については、「[AWS クラウドセキュリティ](#)」を参照してください。インフラストラクチャセキュリティのベストプラクティスを使用して AWS 環境を設計するには、「セキュリティの柱 - AWS Well-Architected Framework」の「[インフラストラクチャ保護](#)」を参照してください。

AWS 公開版 API コールを使用して、ネットワーク経由でアクセスします。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 および TLS 1.3 をお勧めします。
- DHE (Ephemeral Diffie-Hellman) や ECDHE (Elliptic Curve Ephemeral Diffie-Hellman) などの Perfect Forward Secrecy (PFS) を使用した暗号スイートです。これらのモードは、Java 7 以降など、最近のほとんどのシステムでサポートされています。

また、リクエストは、アクセスキー ID と、IAM プリンシパルに関連付けられているシークレットアクセスキーを使用して署名する必要があります。または、[AWS Security Token Service \(AWS STS\)](#) を使用して、一時的なセキュリティ認証情報を生成し、リクエストに署名することもできます。

AWS IoT Events のクォータ

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 に対するデフォルトのクォータ値が記述されています。特に指定がない限り、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。

Service Quotas 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

- デイ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。
- デイ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

ディ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 リソースをさまざまな方法で分類できます。例えば、目的、所有者、環境などです。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、のタグエディタは、タグを作成および管理するための AWS Management Console の中央の統一された方法を提供します。詳細については、「[の操作 AWS Management Console](#)」の「[タグエディタの操作](#)」を参照してください。

また、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。

[AWSタグ付け戦略](#)で追加情報を利用できます。

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
- タグのキーと値は大文字と小文字が区別されます。
- タグの名前または値に "aws:" プレフィックスを使用しないでください。これは、AWS での使用のために予約されているためです。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、特定の連続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=prod」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確認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。

AWS IoT Events のトラブルシューティング

これらのセクションの情報を使用して、AWS IoT Events に関する問題のトラブルシューティングと解決を行います。

トピック

- [一般的な AWS IoT Events 問題と解決策](#)
- [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](#)

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、の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、「[ディテクターモデルの制限と制限](#)」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 ([UpdateDetector「モデル」](#)を参照)、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、ディテクターが間違っただけの状態になります。

ソリューション

[BatchPutメッセージ](#) 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、('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が、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

と UpdateDetectorModel APIs を呼び出そう CreateDetectorModel と InvalidRequestException と、 が表示されます。

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、[CreateDetector「Model」](#)と[UpdateDetector「Model」](#)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 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

[SetTimerアクション](#) アクションを参照して、正しいパラメータを指定し、タイマーを再度設定します。

詳細については、「[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](#)」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 はまず、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てくださいJSON。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 AWS IoT Events 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、「[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 がない。

- 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

例えば、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、[the section called “ナビゲーションペインへの入力の作成”](#)「」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。

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

AWS IoT Events は、ディテクターモデルに入力データを送信せずに、ディテクターモデルを分析して分析結果を生成できます。AWS IoT Events は、このセクションで説明する一連の分析を実行して、ディテクターモデルをチェックします。このアドバンス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、[ディテクターモデルの分析と診断情報](#) 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、[ディテクターモデルの分析 \(コンソール\)](#) または [ディテクターモデルの分析 \(AWS CLI\)](#) を参照してください。

トピック

- [ディテクターモデルの分析と診断情報](#)
- [ディテクターモデルの分析 \(コンソール\)](#)
- [ディテクターモデルの分析 \(AWS CLI\)](#)

ディテクターモデルの分析と診断情報

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ると、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、「[表現](#)」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。

ディテクターモデルエラーのトラブルシューティング

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、[サポートされているアクション](#) 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、[サポートされているアクション](#) を参照してください。

- Message: DetectorModel definition には *aws-service* アクションがありますが、この *aws-service* サービスは region *region-name* では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ても AWS IoT Events、現在のリージョンでそのアクションを使用できないと、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に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せん。

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、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: *transition-event-name* TransitionEventは、存在しない *state-name* に移行します。

解決策: が移行イベント AWS IoT Events が参照した次の状態を見つけられない場合、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： に共有状態名 `DetectorModelDefinition` がありました。状態の数の繰り返しを含む `state-name` 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 `initialStateName #####` は、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、受信データは評価 AWS IoT Events されません。

- メッセージ： で設定できるのは、秒と `durationExpression` の 1 つだけです `SetTimer`。

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 [SetTimer「アクション」](#) 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 まず、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 [SetVariable](#)」の「[アクション](#)」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、「[構文](#)」を参照してください。

- メッセージ: IotSitetwiseAction 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 AWS IoT Events の *fieldName* を解析できなかった場合、このエラーが表示されることがあります IotSitetwiseAction。 *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、API リファレンス [IotSiteWiseAction](#) の「」を参照してください。 AWS IoT Events

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 がタイマーの継続時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、「[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: が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、「[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。

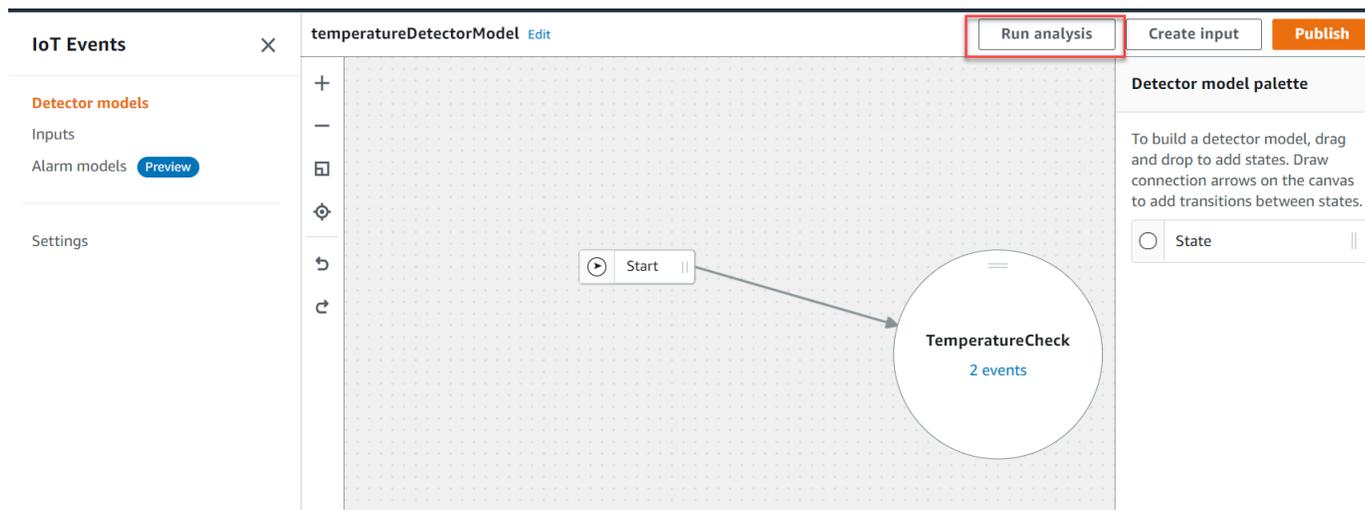
- メッセージ: デテクターモデル定義に無効な が含まれています InputName: *input-name*

解決策: デ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)。

ディテクターモデルの分析 (コンソール)

次のステップでは、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。

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。



以下は、AWS IoT Events コンソールでの分析結果の例です。

The screenshot shows the AWS IoT Events console interface for editing a detector model named 'temperatureDetectorModel'. On the left, there is a navigation menu with 'Detector models', 'Inputs', 'Alarm models', and 'Settings'. The main canvas displays a state machine diagram with a 'Start' button and a state named 'TemperatureCheck' containing '2 events'. On the right, the 'Detector model palette' shows a 'State' component. At the bottom, a 'Detector model analysis' panel is highlighted with a red box, showing a summary of results: (1) All, (0) Error, (0) Warning, and (1) Information. The information message states: 'Message: Inferred data types [Integer] for \$variable.temperatureChecked'.

Note

AWS IoT Events がディテクターモデルの分析をスタートしてから、分析結果を取得するのに最大 24 時間かかります。

ディテクターモデルの分析 (AWS CLI)

次のステップでは、AWS CLI を使用してディテクターモデルを分析します。

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"  
}  
}
```

AWS CLI を使用して既存のディテクターモデルを分析する場合、は次のいずれかを選択してディテクターモデルの定義を取得します。

- AWS IoT Events コンソールを使用する場合は、次の手順を実行します。
 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。
- AWS CLI を使用する場合は、次のようにします。
 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"
}
```

- 出力から分析 ID をコピーします。
- 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 デ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 - AWS IoT Events は、デテクターモデルの分析を終了しました。
 - FAILED - AWS IoT Events は、デ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。
- 次のコマンドを実行して、デテクターモデルの 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

AWS IoT Events がディテクターモデルの分析をスタートしてから、分析結果を取得するのに最大 24 時間かかります。

AWS IoT Events コマンド

この章では、サポートされている ウェブサービス プロトコルのサンプルリクエスト、レスポンス、エラーなど、AWS IoT Events のすべての API オペレーションについて詳しく説明します。

AWS IoT Events アクション

AWS IoT Events API コマンドを使用して、入力およびディテクターモデルの生成、読み取り、更新、削除、およびバージョン一覧の取得ができます。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の[AWS IoT Events セクション](#)には、AWS IoT Events の管理と操作に使用できる AWS CLI コマンドが含まれています。

AWS IoT Events データ

AWS IoT Events データ API コマンドを使用して、ディテクターに入力を送信したり、ディテクターを一覧表示したり、ディテクターのステータスを表示または更新したりできます。詳細については、AWS IoT Events API リファレンスの AWS IoT Events データ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の[AWS IoT Events 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。

ドキュメント履歴

次の表は、2020年9月17日より後の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
リージョン起動	AWS IoT Events が、アジアパシフィック (ムンバイ) リージョンで使用できるようになりました。	2021年9月30日
リージョン起動	AWS IoT Events は現在、AWS GovCloud (米国西部) リージョンで利用可能です。	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 は、ディテクターモデルを分析し、トラブルシューティングの役に立つ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) で AWS IoT Events を起動。	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示しています。

変更	説明	日付
追加	タイプ検証を リファレンス に追加。	2020年8月3日
追加	リージョン情報を 他の AWS サービスの使用 に追加。	2020年5月7日
追加、更新	ペイロード カスタマイズ特徴、新しいイベントアクションを追加: Amazon DynamoDB と AWS IoT SiteWise。	2020年4月27日
編集	ステートマシンの概念の新しい説明を追加しました。コンテンツの一般的な編集。	2019年10月31日
追加	ディテクターモデルの条件表現に新しい組み込み関数を追加。	2019年9月10日
追加	ディテクターモデルの例を追加。	2019年8月5日
追加	新しいイベントアクションを追加しました: Lambda、Amazon SQS、Kinesis Data Firehose、AWS IoT Events 入力。	2019年7月19日
追加、修正	timeout() 関数の正しい説明。休止状態アカウントに関	2019年6月11日

変更	説明	日付
	するベストプラクティスを追加。	
訂正	更新: コンソールデバッグオプションページのイメージ。コンソール許可ポリシー。	2019年6月5日
更新	AWS IoT Events サービスが一般共用開始。	2019年5月30日
追加、更新	セキュリティ情報を更新。注釈付きディテクターモデルの例を追加。	2019年5月22日
訂正	更新: 限定プレビューダウンロードへのリンク、Amazon SNS ペイロードの例、CreateDetectorModel に必要な許可の追加。	2019年5月17日
追加	セキュリティに関する情報を追加。	2019年5月9日
訂正	限定プレビューのダウンロードへのリンクを修正。	2019年4月19日
編集	編集上の改訂。	2019年4月16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
編集	編集上の改訂。	2018年5月18日

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