aws

Entwicklerhandbuch

AWS IoT Events

AWS IoT Events: Entwicklerhandbuch

Table of Contents

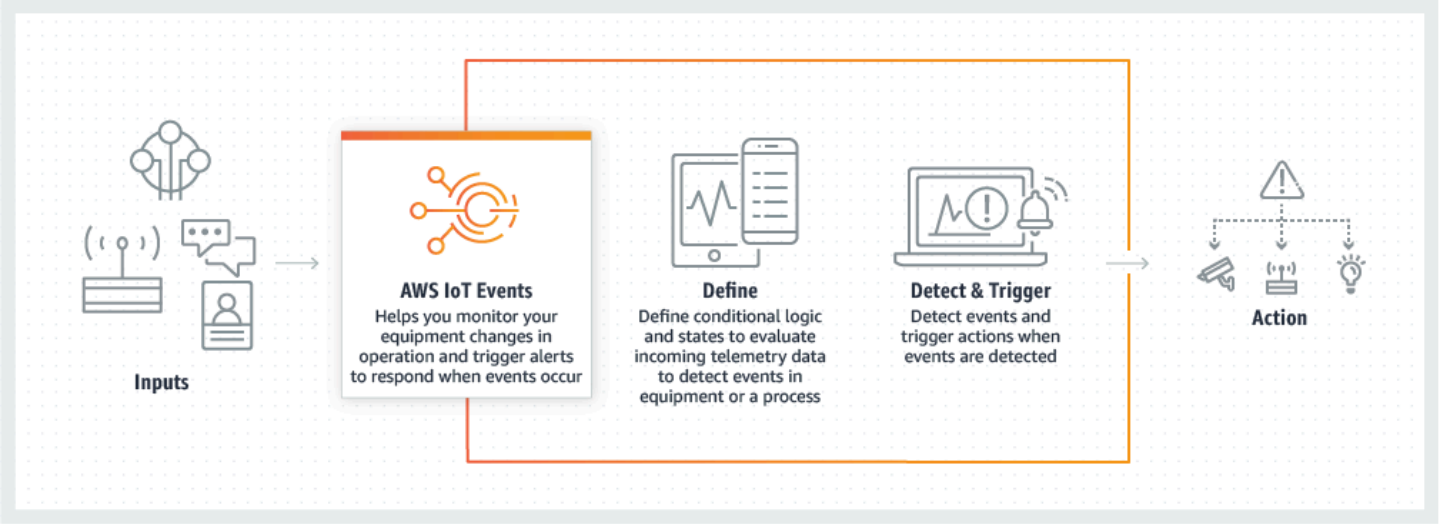
Hinweis zum Ende des Supports: Am 20. Mai 2026 AWS endet der Support für AWS IoT Events. Nach dem 20. Mai 2026 können Sie nicht mehr auf die AWS IoT Events Konsole oder AWS IoT Events die Ressourcen zugreifen. Weitere Informationen finden Sie unter AWS IoT Events Ende des Supports.

Die vorliegende Übersetzung wurde maschinell erstellt. Im Falle eines Konflikts oder eines Widerspruchs zwischen dieser übersetzten Fassung und der englischen Fassung (einschließlich infolge von Verzögerungen bei der Übersetzung) ist die englische Fassung maßgeblich.

Was ist AWS IoT Events?

AWS IoT Events ermöglicht es Ihnen, Ihre Geräte oder Geräteflotten auf Ausfälle oder Betriebsänderungen zu überwachen und bei Auftreten solcher Ereignisse Maßnahmen auszulösen. AWS IoT Events überwacht kontinuierlich IoT-Sensordaten von Geräten, Prozessen, Anwendungen und anderen AWS Diensten, um wichtige Ereignisse zu identifizieren, damit Sie Maßnahmen ergreifen können.

Verwenden Sie diese Option AWS IoT Events , um komplexe Anwendungen zur Ereignisüberwachung in der AWS Cloud zu erstellen, auf die Sie über die AWS IoT Events Konsole oder APIs zugreifen können.



Topics

- Vorteile und Funktionen
- Anwendungsfälle

Vorteile und Funktionen

Akzeptieren Sie Eingaben aus mehreren Quellen

AWS IoT Events akzeptiert Eingaben aus vielen IoT-Telemetriedatenquellen. Dazu gehören Sensorgeräte, Verwaltungsanwendungen und andere AWS IoT Dienste wie AWS IoT Core und AWS IoT Analytics. Sie können jede Telemetriedateneingabe über eine Standard-API-Schnittstelle (`BatchPutMessage`API) oder die AWS IoT Events Konsole AWS IoT Events per Push übertragen.

Weitere Informationen zu den ersten Schritten mit finden Sie AWS IoT Events unterErste Schritte mit der AWS IoT Events Konsole.

Verwenden Sie einfache logische Ausdrücke, um komplexe Ereignismuster zu erkennen

AWS IoT Events kann Muster von Ereignissen erkennen, die mehrere Eingaben von einem einzelnen IoT-Gerät oder einer einzelnen IoT-Anwendung oder von verschiedenen Geräten und vielen unabhängigen Sensoren beinhalten. Dies ist besonders nützlich, da jeder Sensor und jede Anwendung wichtige Informationen liefert. Aber nur durch die Kombination verschiedener Sensor- und Anwendungsdaten können Sie sich ein vollständiges Bild von der Leistung und Qualität der Abläufe machen. Sie können AWS IoT Events Melder so konfigurieren, dass sie diese Ereignisse mithilfe einfacher logischer Ausdrücke anstelle von komplexem Code erkennen.

Weitere Hinweise zu logischen Ausdrücken finden Sie unterAusdrücke zum Filtern, Transformieren und Verarbeiten von Ereignisdaten.

Auf Ereignissen basierende Aktionen auslösen

AWS IoT Events ermöglicht es Ihnen, Aktionen in Amazon Simple Notification Service (Amazon SNS), Lambda AWS IoT Core, Amazon SQS und Amazon Kinesis Firehose direkt auszulösen. Sie können eine AWS Lambda Funktion auch mithilfe der AWS IoT Regel-Engine auslösen, sodass Sie Aktionen mithilfe anderer Dienste wie Connect Customer oder Ihrer eigenen ERP-Anwendungen (Enterprise Resource Planning) ausführen können.

AWS IoT Events enthält eine vorgefertigte Bibliothek mit Aktionen, die Sie ausführen können, und ermöglicht es Ihnen auch, Ihre eigenen Aktionen zu definieren.

Weitere Informationen zum Auslösen von Aktionen auf der Grundlage von Ereignissen finden Sie unter. Unterstützte Aktionen zum Empfangen von Daten und Auslösen von Aktionen in AWS IoT Events

Automatisch skalieren, um den Anforderungen Ihrer Flotte gerecht zu werden

AWS IoT Events skaliert automatisch, wenn Sie homogene Geräte anschließen. Sie können einen Detektor einmal für einen bestimmten Gerätetyp definieren, und der Dienst skaliert und verwaltet automatisch alle Instanzen dieses Geräts, mit denen eine Verbindung hergestellt wird AWS IoT Events.

Beispiele für Detektormodelle finden Sie unterAWS IoT Events Beispiele für Detektormodelle.

Anwendungsfälle

AWS IoT Events hat viele Verwendungsmöglichkeiten. Hier sind ein paar Beispiele für Anwendungsfälle.

Überwachen und warten Sie Remote-Geräte

Die Überwachung einer Flotte von ferngesteuerten Maschinen kann eine Herausforderung sein, insbesondere wenn eine Fehlfunktion ohne klaren Kontext auftritt. Wenn eine Maschine nicht mehr funktioniert, kann dies bedeuten, dass die gesamte Verarbeitungseinheit oder Maschine ausgetauscht werden muss. Aber das ist nicht nachhaltig. Damit können AWS IoT Events Sie Nachrichten von mehreren Sensoren auf jedem Gerät empfangen, um bestimmte Probleme im Laufe der Zeit zu diagnostizieren. Anstatt die gesamte Einheit auszutauschen, verfügen Sie jetzt über die erforderlichen Informationen, um einen Techniker mit dem genauen Teil zu beauftragen, der ausgetauscht werden muss. Bei Millionen von Maschinen können sich die Einsparungen auf Millionen von Dollar summieren, wodurch Ihre Gesamtkosten für den Besitz oder die Wartung jeder Maschine sinken.

Industrieroboter verwalten

Der Einsatz von Robotern in Ihren Einrichtungen zur Automatisierung der Paketbewegung kann die Effizienz erheblich steigern. Um die Kosten zu minimieren, können Roboter mit einfachen, kostengünstigen Sensoren ausgestattet werden, die Daten an die Cloud melden. Bei Dutzenden von Sensoren und Hunderten von Betriebsmodi kann es jedoch schwierig sein, Probleme in Echtzeit zu erkennen. Damit können Sie ein Expertensystem aufbauen AWS IoT Events, das diese Sensordaten in der Cloud verarbeitet und Warnmeldungen erstellt, um das technische Personal automatisch zu benachrichtigen, wenn ein Ausfall unmittelbar bevorsteht.

Verfolgen Sie Gebäudeautomationssysteme

In Rechenzentren trägt die Überwachung auf hohe Temperaturen und niedrige Luftfeuchtigkeit dazu bei, Geräteausfälle zu verhindern. Sensoren werden häufig von vielen Herstellern gekauft und jeder Typ wird mit einer eigenen Verwaltungssoftware geliefert. Verwaltungssoftware verschiedener Anbieter ist jedoch manchmal nicht kompatibel, was die Erkennung von Problemen erschwert. Mithilfe dieser Funktion können Sie Warnmeldungen einrichten AWS IoT Events, um Ihre Betriebsanalysten rechtzeitig vor einem Ausfall über Probleme mit Ihren Heiz- und Kühlsystemen zu informieren. Auf diese Weise können Sie eine ungeplante Abschaltung des Rechenzentrums verhindern, die Tausende von Dollar für den Austausch von Geräten und potenzielle Umsatzeinbußen kosten würde.

AWS IoT Events Ende des Supports

Nach reiflicher Überlegung haben wir beschlossen, den Support für den AWS IoT Events Service mit Wirkung zum 20. Mai 2026 einzustellen. AWS IoT Events wird ab dem 20. Mai 2025 keine neuen Kunden mehr annehmen. Als Bestandskunde mit einem Konto, das vor dem 20. Mai 2025 für den Service registriert wurde, können Sie die AWS IoT Events Funktionen weiterhin nutzen. Nach dem 20. Mai 2026 können Sie ihn nicht mehr nutzen AWS IoT Events.

Auf dieser Seite finden Sie Anleitungen und Hinweise für AWS IoT Events Kunden, wie sie auf eine alternative Lösung umsteigen können, die Ihren Geschäftsanforderungen entspricht.

> **Note**
>
> Die in diesen Leitfäden vorgestellten Lösungen sollen als anschauliche Beispiele dienen und nicht als produktionsreifer Ersatz für Funktionen dienen. AWS IoT Events Passen Sie den Code, den Workflow und die zugehörigen AWS Ressourcen an Ihre Geschäftsanforderungen an.

Themen

Überlegungen bei der Migration weg von AWS IoT Events

- Implementieren Sie bewährte Sicherheitsmethoden, einschließlich der Verwendung von IAM-Rollen mit den geringsten Rechten für jede Komponente und der Verschlüsselung von Daten im Ruhezustand und bei der Übertragung. Weitere Informationen finden Sie unter Bewährte Methoden für die Sicherheit in IAM im IAM-Benutzerhandbuch.
- Berücksichtigen Sie die Anzahl der Shards für den Kinesis-Stream auf der Grundlage Ihrer Datenaufnahmeanforderungen. Weitere Informationen zu Kinesis-Shards finden Sie in der Terminologie und den Konzepten von Amazon Kinesis Data Streams im Amazon Kinesis Data Streams Developer Guide.

- Richten Sie mithilfe von Metriken und Protokollen umfassende Überwachungs- und CloudWatch Debugging-Funktionen ein. Weitere Informationen finden Sie unter Was ist CloudWatch? im CloudWatch Amazon-Benutzerhandbuch.
- Berücksichtigen Sie die Struktur Ihrer Fehlerbehandlung, einschließlich der Verwaltung von Nachrichten, deren Verarbeitung wiederholt fehlschlägt, der Implementierung von Richtlinien für Wiederholungsversuche und der Einrichtung eines Prozesses zur Isolierung und Analyse problematischer Nachrichten.
- Verwenden Sie den AWS Preisrechner, um die Kosten für Ihren speziellen Anwendungsfall zu schätzen.

Migrationsverfahren für Detektormodelle in AWS IoT Events

In diesem Abschnitt werden alternative Lösungen beschrieben, die ähnliche Funktionen des Detektormodells bieten, wenn Sie von wechseln AWS IoT Events.

Sie können die Datenaufnahme über AWS IoT Core Regeln auf eine Kombination anderer AWS Dienste migrieren. Anstatt die Daten über die BatchPutMessageAPI aufzunehmen, können die Daten an das MQTT-Thema weitergeleitet werden. AWS IoT Core

Dieser Migrationsansatz nutzt AWS IoT Core MQTT-Themen als Einstiegspunkt für Ihre IoT-Daten und ersetzt die direkte Eingabe zu. AWS IoT Events MQTT-Themen werden aus mehreren wichtigen Gründen ausgewählt. Sie bieten aufgrund der weit verbreiteten Verwendung von MQTT in der Branche eine breite Kompatibilität mit IoT-Geräten. Diese Themen können große Mengen an Nachrichten von zahlreichen Geräten verarbeiten und gewährleisten so die Skalierbarkeit. Sie bieten auch Flexibilität beim Routing und Filtern von Nachrichten nach Inhalt oder Gerätetyp. Darüber hinaus lassen sich AWS IoT Core MQTT-Themen nahtlos in andere AWS Dienste integrieren, was den Migrationsprozess erleichtert.

Daten fließen von MQTT-Themen in eine Architektur, die Amazon Kinesis Data Streams, eine AWS Lambda Funktion, eine Amazon DynamoDB-Tabelle und Amazon-Zeitpläne kombiniert. EventBridge Diese Kombination von Diensten repliziert und erweitert die Funktionen, die zuvor von bereitgestellt wurden AWS IoT Events, und bietet Ihnen mehr Flexibilität und Kontrolle über Ihre IoT-Datenverarbeitungspipeline.

Architekturen im Vergleich

Die aktuelle AWS IoT Events Architektur nimmt Daten über eine AWS IoT Core Regel und die `BatchPutMessage` API auf. Diese Architektur wird AWS IoT Core für die Datenaufnahme und die

Veröffentlichung von Ereignissen verwendet, wobei Nachrichten über AWS IoT Events Eingänge an Detektormodelle weitergeleitet werden, die die Zustandslogik definieren. Eine IAM-Rolle verwaltet die erforderlichen Berechtigungen.

Die neue Lösung kümmert sich AWS IoT Core um die Datenaufnahme (jetzt mit speziellen MQTT-Themen für Eingabe und Ausgabe). Es führt Kinesis Data Streams für die Datenpartitionierung und eine Evaluator-Lambda-Funktion für die Zustandslogik ein. Gerätestatus werden jetzt in einer DynamoDB-Tabelle gespeichert, und eine erweiterte IAM-Rolle verwaltet die Berechtigungen für diese Dienste.

Zweck	Lösung	Unterschiede
Datenaufnahme — Empfängt Daten von IoT-Geräten	AWS IoT Core	Erfordert jetzt zwei unterschiedliche MQTT-Themen: eines für die Erfassung von Gerätedaten und eines für die Veröffentlichung von Ausgabeereignissen
Nachrichtenrichtung — Leitet eingehende Nachrichten an entsprechende Dienste weiter	AWS IoT Core Regel für die Nachrichtenweiterleitung	Behält die gleiche Routing-Funktionalität bei, leitet aber Nachrichten jetzt an Kinesis Data Streams weiter, anstatt AWS IoT Events
Datenverarbeitung — Verarbeitet und organisiert eingehende Datenströme	Kinesis Data Streams	Ersetzt die AWS IoT Events Eingabefunktionalität und ermöglicht die Datenaufnahme durch eine Geräte-ID-Partitionierung für die Nachrichtenverarbeitung
Logikauswertung — Verarbeitet Statusänderungen und löst Aktionen aus	Evaluator Lambda	Ersetzt das AWS IoT Events Detektormodell und ermöglicht eine anpassbare Auswertung der Zustandslogik durch Code anstelle eines visuellen Workflows
Statusverwaltung — Behält den Gerätestatus bei	DynamoDB-Tabelle	Neue Komponente, die die persistente Speicherung von Gerätestatus ermöglicht und die interne AWS IoT Events Statusverwaltung ersetzt

Zweck	Lösung	Unterschiede
Sicherheit — Verwaltet Dienstber echtigungen	IAM role (IAM-Rolle)	Die aktualisierten Berechtigungen umfassen jetzt zusätzlich zu den vorhandenen Berechtig ungen den Zugriff auf Kinesis Data Streams und EventBridge DynamoDB AWS IoT Core

Schritt 1: (Optional) Exportieren AWS IoT Events Sie die Modellkonfigurationen der Detektoren

Bevor Sie neue Ressourcen erstellen, exportieren Sie Ihre AWS IoT Events Detektormodelldefinitionen. Diese enthalten Ihre Logik zur Ereignisverarbeitung und können als historische Referenz für die Implementierung Ihrer neuen Lösung dienen.

Console

Führen Sie mithilfe von die folgenden Schritte aus AWS IoT Events AWS-Managementkonsole, um Ihre Meldermodellkonfigurationen zu exportieren:

Um Detektormodelle zu exportieren, verwenden Sie AWS-Managementkonsole

1. Melden Sie sich an der AWS IoT Events -Konsole an.
2. Wählen Sie im linken Navigationsbereich Detector models (Detektormodelle).
3. Wählen Sie das zu exportierende Detektormodell aus.
4. Wählen Sie Export aus. Lesen Sie die Informationsmeldung zur Ausgabe und wählen Sie dann erneut Exportieren.
5. Wiederholen Sie den Vorgang für jedes Meldermodell, das Sie exportieren möchten.

Eine Datei, die eine JSON-Ausgabe Ihres Detektormodells enthält, wird dem Download-Ordner Ihres Browsers hinzugefügt. Sie können optional jede Konfiguration des Detektormodells speichern, um historische Daten beizubehalten.

AWS CLI

Führen Sie mit dem die folgenden Befehle aus AWS CLI, um Ihre Detektormodellkonfigurationen zu exportieren:

Um Detektormodelle zu exportieren mit AWS CLI

1. Alle Meldermodelle in Ihrem Konto auflisten:

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

2. Exportieren Sie für jedes Meldermodell dessen Konfiguration, indem Sie den folgenden Befehl ausführen:

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

3. Speichern Sie die Ausgabe für jedes Detektormodell.

Schritt 2: Erstellen einer IAM-Rolle

Erstellen Sie eine IAM-Rolle, um Berechtigungen zur Replikation der Funktionalität von bereitzustellen. AWS IoT Events Die Rolle in diesem Beispiel gewährt Zugriff auf DynamoDB für die Statusverwaltung, EventBridge für die Planung, Kinesis Data Streams für die Datenaufnahme, für die Veröffentlichung von Nachrichten und AWS IoT Core für die Protokollierung. CloudWatch Zusammen können diese Dienste als Ersatz für verwendet werden. AWS IoT Events

1. Erstellen Sie eine IAM-Rolle mit den folgenden Berechtigungen. Ausführlichere Anweisungen zum Erstellen einer IAM-Rolle finden Sie unter Erstellen einer Rolle zum Delegieren von Berechtigungen für einen AWS Dienst im IAM-Benutzerhandbuch.

   JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "DynamoDBAccess",
            "Effect": "Allow",
            "Action": [
                "dynamodb:GetItem",
                "dynamodb:PutItem",
                "dynamodb:UpdateItem",
                "dynamodb:DeleteItem",
                "dynamodb:Query",
```

```
                "dynamodb:Scan"
            ],
            "Resource": "arn:aws:dynamodb:us-east-1:123456789012:table/
EventsStateTable"
        },
        {
            "Sid": "SchedulerAccess",
            "Effect": "Allow",
            "Action": [
                "scheduler:CreateSchedule",
                "scheduler:DeleteSchedule"
            ],
            "Resource": "arn:aws:scheduler:us-east-1:123456789012:schedule/*"
        },
        {
            "Sid": "KinesisAccess",
            "Effect": "Allow",
            "Action": [
                "kinesis:GetRecords",
                "kinesis:GetShardIterator",
                "kinesis:DescribeStream",
                "kinesis:ListStreams"
            ],
            "Resource": "arn:aws:kinesis:us-east-1:123456789012:stream/*"
        },
        {
            "Sid": "IoTPublishAccess",
            "Effect": "Allow",
            "Action": "iot:Publish",
            "Resource": "arn:aws:iot:us-east-1:123456789012:topic/*"
        },
        {
            "Effect": "Allow",
            "Action": "logs:CreateLogGroup",
            "Resource": "arn:aws:logs:us-east-1:123456789012:*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogStream",
                "logs:PutLogEvents"
            ],
            "Resource": [
```

```
            "arn:aws:logs:us-east-1:123456789012:log-group:/aws/lambda/your-
lambda:*"
            ]
        }
    ]
}
```

2. Fügen Sie die folgende Vertrauensrichtlinie für IAM-Rollen hinzu. Eine Vertrauensrichtlinie ermöglicht es den angegebenen AWS Diensten, die IAM-Rolle zu übernehmen, sodass sie die erforderlichen Aktionen ausführen können. Ausführlichere Anweisungen zur Erstellung einer IAM-Vertrauensrichtlinie finden Sie im IAM-Benutzerhandbuch unter Erstellen einer Rolle mithilfe benutzerdefinierter Vertrauensrichtlinien.

   JSON

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

Schritt 3: Amazon Kinesis Data Streams erstellen

Erstellen Sie Amazon Kinesis Data Streams mit dem AWS-Managementkonsole oder AWS CLI.

Console

Um einen Kinesis-Datenstream mit dem zu erstellen AWS-Managementkonsole, folgen Sie dem Verfahren auf der Seite Einen Datenstream erstellen im Amazon Kinesis Data Streams Developer Guide.

Passen Sie die Anzahl der Shards an Ihre Geräteanzahl und die Größe der Nachrichtennutzlast an.

AWS CLI

Erstellen Sie mithilfe von Amazon Kinesis Data Streams AWS CLI, um die Daten von Ihren Geräten aufzunehmen und zu partitionieren.

Kinesis Data Streams werden bei dieser Migration verwendet, um die Datenaufnahmefunktion von zu ersetzen. AWS IoT Events Es bietet eine skalierbare und effiziente Methode zur Erfassung, Verarbeitung und Analyse von Echtzeit-Streaming-Daten von Ihren IoT-Geräten und bietet gleichzeitig eine flexible Datenverarbeitung und Integration mit anderen AWS Diensten.

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

Passen Sie die Anzahl der Shards an die Anzahl Ihrer Geräte und die Größe der Nachrichtennutzlast an.

Schritt 4: Erstellen oder aktualisieren Sie die MQTT-Nachrichtenrouting-Regel

Sie können eine neue MQTT-Nachrichtenrouting-Regel erstellen oder eine bestehende Regel aktualisieren.

Console

1. Stellen Sie fest, ob Sie eine neue MQTT-Nachrichtenrouting-Regel benötigen oder ob Sie eine bestehende Regel aktualisieren können.
2. Öffnen Sie die AWS IoT Core -Konsole.
3. Wählen Sie im Navigationsbereich Message Routing und dann Regeln aus.

4. Wählen Sie im Abschnitt Verwalten die Option Nachrichtenweiterleitung und dann Regeln aus.
5. Wählen Sie Regel erstellen aus.
6. Geben Sie auf der Seite Regeleigenschaften angeben den AWS IoT Core Regelnamen für Regelname ein. Geben Sie unter Regelbeschreibung — optional eine Beschreibung ein, um zu kennzeichnen, dass Sie Ereignisse verarbeiten und an Kinesis Data Streams weiterleiten.
7. Geben Sie auf der Seite „SQL-Anweisung konfigurieren“ Folgendes für die SQL-Anweisung ein: **`SELECT * FROM 'your-database'`** und wählen Sie dann Weiter.
8. Wählen Sie auf der Seite „Regeln anhängen“ und unter Regelaktionen die Option Kinesis aus.
9. Wählen Sie Ihren Kinesis-Stream für den Stream aus. Geben Sie als Partitionsschlüssel **`your-instance-id`** ein. Wählen Sie die entsprechende Rolle für die IAM-Rolle aus und klicken Sie dann auf Regelaktion hinzufügen.

Weitere Informationen finden Sie unter AWS IoT-Regeln erstellen, um Gerätedaten an andere Dienste weiterzuleiten.

AWS CLI

1. Erstellen Sie eine JSON Datei mit dem Namen und dem folgenden Inhalt. Diese JSON-Konfigurationsdatei definiert eine AWS IoT Core Regel, die alle Nachrichten aus einem Thema auswählt und sie an den angegebenen Kinesis-Stream weiterleitet, wobei die Instanz-ID als Partitionsschlüssel verwendet wird.

```
{
    "sql": "SELECT * FROM 'your-config-file'",
    "description": "Rule to process events and forward to Kinesis Data Streams",
    "actions": [
        {
            "kinesis": {
                "streamName": "your-kinesis-stream-name",
                "roleArn": "arn:aws:iam::your-account-id:role/service-role/your-
iam-role",
                "partitionKey": "${your-instance-id}"
            }
        }
    ],
    "ruleDisabled": false,
    "awsIotSqlVersion": "2016-03-23"
```

```
}
```

2. Erstellen Sie die MQTT-Themenregel mit dem. AWS CLI In diesem Schritt wird mithilfe der AWS CLI eine AWS IoT Core Themenregel unter Verwendung der in der `events_rule.json` Datei definierten Konfiguration erstellt.

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

Schritt 5: Ermitteln Sie den Endpunkt für das MQTT-Zielthema

Verwenden Sie das Ziel-MQTT-Thema, um zu konfigurieren, wo Ihre Themen ausgehende Nachrichten veröffentlichen, und ersetzen Sie damit die Funktionalität, die zuvor von verwendet wurde. AWS IoT Events Der Endpunkt ist für Ihr AWS Konto und Ihre Region einzigartig.

Console

1. Öffnen Sie die AWS IoT Core -Konsole.
2. Wählen Sie im linken Navigationsbereich im Bereich Connect die Option Domain-Konfiguration aus.
3. Wählen Sie die Domainkonfiguration IoT:Data-ATS, um die Detailseite der Konfiguration zu öffnen.
4. Kopieren Sie den Wert für den Domainnamen. Dieser Wert ist der Endpunkt. Speichern Sie den Endpunktwert, da Sie ihn in späteren Schritten benötigen werden.

AWS CLI

Führen Sie den folgenden Befehl aus, um den AWS IoT Core Endpunkt für die Veröffentlichung ausgehender Nachrichten für Ihr Konto zu ermitteln.

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

Schritt 6: Erstellen Sie eine Amazon DynamoDB-Tabelle

Eine Amazon DynamoDB-Tabelle ersetzt die Statusverwaltungsfunktion von und bietet eine skalierbare und flexible Möglichkeit AWS IoT Events, den Status Ihrer Geräte und die Detektormodelllogik in Ihrer neuen Lösungsarchitektur beizubehalten und zu verwalten.

Console

Erstellen Sie eine Amazon DynamoDB-Tabelle, um den Status der Detektormodelle beizubehalten. Weitere Informationen finden Sie unter Erstellen einer Tabelle in DynamoDB im Amazon DynamoDB Developer Guide.

Verwenden Sie Folgendes für die Tabellendetails:

- Geben Sie unter Tabellenname einen Tabellennamen Ihrer Wahl ein.
- Geben Sie für Partitionsschlüssel Ihre eigene Instanz-ID ein.
- Sie können die Standardeinstellungen für die Tabelleneinstellungen verwenden

AWS CLI

Führen Sie den folgenden Befehl aus, um eine DynamoDB-Tabelle zu erstellen.

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

Schritt 7: Erstellen Sie eine AWS Lambda Funktion (Konsole)

Die Lambda-Funktion dient als zentrale Verarbeitungs-Engine und ersetzt die Bewertungslogik des Detektormodells von AWS IoT Events. In diesem Beispiel integrieren wir andere AWS Dienste, um eingehende Daten zu verarbeiten, den Status zu verwalten und Aktionen auf der Grundlage Ihrer definierten Regeln auszulösen.

Erstellen Sie eine Lambda-Funktion mit NodeJS Laufzeit. Verwenden Sie den folgenden Codeausschnitt, der die hartcodierten Konstanten ersetzt:

1. Öffnen Sie die AWS Lambda console.

2. Wählen Sie Funktion erstellen.
3. Geben Sie einen Namen für den Funktionsnamen ein.
4. Wählen Sie NodeJS 2.x als Runtime aus.
5. Wählen Sie in der Dropdownliste Standardausführungsrolle ändern die Option Bestehende Rolle verwenden und wählen Sie dann die IAM-Rolle aus, die Sie in den vorherigen Schritten erstellt haben.
6. Wählen Sie Funktion erstellen.
7. Fügen Sie den folgenden Codeausschnitt ein, nachdem Sie die hartcodierten Konstanten ersetzt haben.
8. Fügen Sie nach der Erstellung Ihrer Funktion auf der Registerkarte Code das folgende Codebeispiel ein und ersetzen Sie den **`your-destination-endpoint`** Endpunkt durch Ihren eigenen.

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


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


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

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

    const processedRecords = [];
```

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

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

            const output = await handleDecodedData(decodedData);

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

            processedRecords.push(processedData);

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

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

// Helper function to handle decoded data
async function handleDecodedData(payload) {
```

```
    try {
        // Parse the decoded data
        const parsedData = JSON.parse(payload);

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

        await iotEvents.process(instanceId, inputData)

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


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

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

```
                }
            }));

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

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

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

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

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

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

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

        const scheduleName = `your-schedule-name-${instanceId}-schedule`;
        const scheduleParams = {
            Name: scheduleName,
            FlexibleTimeWindow: {
                Mode: 'OFF'
```

```
            },
            ScheduleExpression: `rate(${frequencyInMinutes} minutes)`,
            Target: {
                Arn: "arn:aws::kinesis:your-region:your-account-id:stream/your-kinesis-
stream-name",
                RoleArn: "arn:aws::iam::your-account-id:role/service-role/your-iam-
role",
                Input: base64Payload,
                KinesisParameters: {
                    PartitionKey: instanceId,
                },
                RetryPolicy: {
                    MaximumRetryAttempts: 3
                }
            },

        };

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

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

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

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

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

```
    }

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


class IoTEvents {

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

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

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

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

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

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

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

class IoTEventsState {
```

```
    constructor() {
        this.eventsList = []
    }

    events(eventListArg) {
        this.eventsList.push(...eventListArg);
        return this;
    }

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

        return currentState;
    }
}

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

let processTimerEvent = new Event(
    (currentState) => {
        const source = currentState.input('source');
        console.log(`Evaluating timer event with source ${source}`);
        const booleanOutput = (source !== undefined && source !== null &&
```

```
            typeof source === 'string' &&
            source.toLowerCase() === 'timer' &&
            // check if the currentState == state from the timer payload
            currentState.input('currentState') !== undefined &&
            currentState.input('currentState') !== null &&
            currentState.input('currentState').toLowerCase !== 'normal');
        console.log(`Timer event evaluated as ${booleanOutput}`);
        return booleanOutput;
    },
    async (currentState) => {
        await currentState.executeAction('MQTT', `{"state": "timer timed out in
 Alarming state" }`);
    }
);

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

Schritt 8: Einen Amazon Kinesis Data Streams Streams-Trigger hinzufügen

Fügen Sie der Lambda-Funktion mithilfe von oder einen Kinesis Data Streams Streams-Trigger hinzu. AWS-Managementkonsole AWS CLI

Durch das Hinzufügen eines Kinesis Data Streams Streams-Triggers zu Ihrer Lambda-Funktion wird die Verbindung zwischen Ihrer Datenerfassungspipeline und Ihrer Verarbeitungslogik hergestellt, sodass eingehende IoT-Datenströme automatisch ausgewertet und in Echtzeit auf Ereignisse reagiert werden kann, ähnlich wie bei der Verarbeitung von Eingaben. AWS IoT Events

Console

Weitere Informationen finden Sie unter Erstellen einer Ereignisquellenzuordnung zum Aufrufen einer Lambda-Funktion im AWS Lambda Entwicklerhandbuch.

Einzelheiten zur Zuordnung der Ereignisquellen finden Sie wie folgt:

- Geben Sie als Funktionsname den Lambda-Namen ein, der in Schritt 7: Erstellen Sie eine AWS Lambda Funktion (Konsole) verwendet wird.
- Geben Sie für Consumer — optional den ARN für Ihren Kinesis-Stream ein.
- Geben Sie für Batch size (Stapelgröße) **`10`** ein.

AWS CLI

Führen Sie den folgenden Befehl aus, um den Lambda-Funktionstrigger zu erstellen.

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

Schritt 9: Testen Sie die Funktionen zur Datenaufnahme und -ausgabe ()AWS CLI

Veröffentlichen Sie eine Nutzlast zum MQTT-Thema, die auf dem basiert, was Sie in Ihrem Detektormodell definiert haben. Im Folgenden finden Sie ein Beispiel für eine Payload zum MQTT-Thema `your-topic-name` zum Testen einer Implementierung.

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

Sie sollten eine MQTT-Nachricht sehen, die zu einem Thema veröffentlicht wurde und den folgenden (oder ähnlichen) Inhalt hat:

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

Migrationsverfahren für AWS IoT SiteWise Alarme in AWS IoT Events

In diesem Abschnitt werden alternative Lösungen beschrieben, die bei der Migration von ähnliche Alarmfunktionen bieten AWS IoT Events.

Für AWS IoT SiteWise Immobilien, die AWS IoT Events Alarme verwenden, können Sie zu einer Lösung migrieren, die CloudWatch Alarme verwendet. Dieser Ansatz bietet robuste Überwachungsfunktionen mit etablierten SLAs und zusätzlichen Funktionen wie der Erkennung von Anomalien und gruppierten Alarmen.

Architekturen im Vergleich

Die aktuelle AWS IoT Events Alarmkonfiguration für AWS IoT SiteWise Eigenschaften muss `AssetModelCompositeModels` im Asset-Modell erstellt werden, wie unter Definieren externer Alarme AWS IoT SiteWise im AWS IoT SiteWise Benutzerhandbuch beschrieben. Änderungen an der neuen Lösung werden in der Regel über die AWS IoT Events Konsole verwaltet.

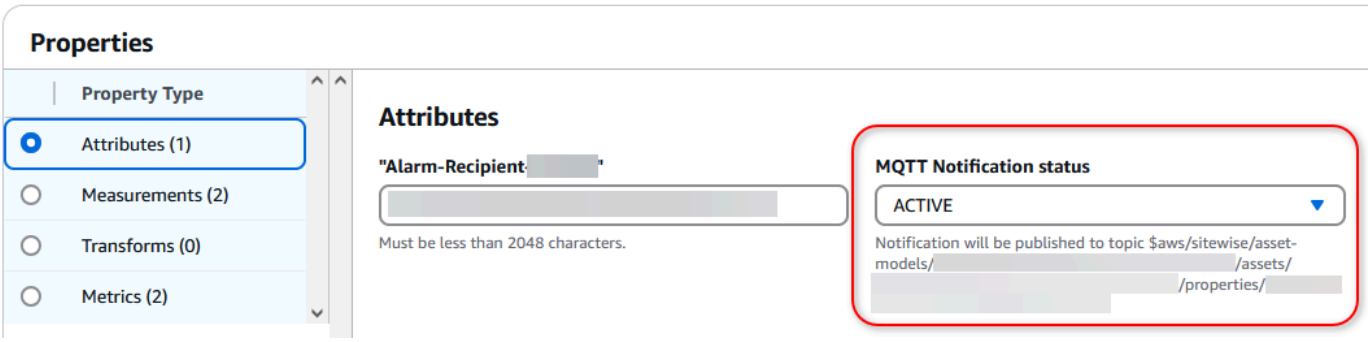
Die neue Lösung ermöglicht das Alarmmanagement mithilfe von CloudWatch Alarmen. Dieser Ansatz verwendet AWS IoT SiteWise Benachrichtigungen, um Eigenschaftsdatenpunkte in AWS IoT Core MQTT-Themen zu veröffentlichen, die dann von einer Lambda-Funktion verarbeitet werden. Die Funktion wandelt diese Benachrichtigungen in CloudWatch Metriken um und ermöglicht so die Alarmüberwachung über CloudWatch das Alarmierungs-Framework.

Zweck	Lösung	Unterschiede
Datenquelle — Immobiliendaten von AWS IoT SiteWise	AWS IoT SiteWise MQTT-Benachrichtigungen	Ersetzt die direkte Integration von IoT Events durch MQTT-Benachrichtigungen von Immobilien AWS IoT SiteWise
Datenverarbeitung — Transformiert Immobiliendaten	Lambda-Funktion	Verarbeitet AWS IoT SiteWise Eigenschaftsbenachrichtigungen und wandelt sie in Metriken um CloudWatch
Alarmauswertung — Überwacht Messwerte und löst Alarme aus	CloudWatch Amazon-Alarme	Ersetzt AWS IoT Events Alarme durch CloudWatch Alarme und bietet zusätzliche Funktionen wie die Erkennung von Anomalien
Integration — Verbindung mit AWS IoT SiteWise	AWS IoT SiteWise externe Alarme	Optionale Möglichkeit, CloudWatch Alarme wieder AWS IoT SiteWise als externe Alarme zu importieren

Schritt 1: Aktivieren Sie MQTT-Benachrichtigungen auf der Asset-Eigenschaft

Wenn Sie AWS IoT Events Integrationen für AWS IoT SiteWise Alarme verwenden, können Sie MQTT-Benachrichtigungen für jede zu überwachende Eigenschaft aktivieren.

1. Folgen Sie der Anleitung Alarme für Anlagen konfigurieren, AWS IoT SiteWise bis Sie den Schritt zum Bearbeiten der Eigenschaften des Asset-Modells abgeschlossen haben.
2. Ändern Sie für jede zu migrierende Eigenschaft den Status der MQTT-Benachrichtigung auf AKTIV.



3. Notieren Sie sich den Themenpfad, unter dem der Alarm für jedes geänderte Alarmattribut veröffentlicht wird.

Weitere Informationen finden Sie in den folgenden Dokumentationsressourcen:

- Erfahren Sie mehr über Asset-Eigenschaften in den MQTT-Themen im AWS IoT SiteWise Benutzerhandbuch.
- MQTT-Themen im AWS IoT Entwicklerhandbuch.

Schritt 2: Erstellen Sie eine Funktion AWS Lambda

Erstellen Sie eine Lambda-Funktion zum Lesen des vom MQTT-Thema veröffentlichten TQV-Arrays und veröffentlichen Sie einzelne Werte in. CloudWatch Wir werden diese Lambda-Funktion als Zielaktion verwenden, die in AWS IoT Core Message Rules ausgelöst werden soll.

1. Öffnen Sie die AWS Lambda console.
2. Wählen Sie Funktion erstellen.
3. Geben Sie einen Namen für den Funktionsnamen ein.
4. Wählen Sie NodeJS 2.x als Runtime aus.
5. Wählen Sie in der Dropdownliste Standardausführungsrolle ändern die Option Bestehende Rolle verwenden und wählen Sie dann die IAM-Rolle aus, die Sie in den vorherigen Schritten erstellt haben.

> **Note**
>
> Bei diesem Verfahren wird davon ausgegangen, dass Sie Ihr Meldermodell bereits migriert haben. Wenn Sie keine IAM-Rolle haben, finden Sie weitere Informationen unter. ???

6. Wählen Sie Funktion erstellen.
7. Fügen Sie den folgenden Codeausschnitt ein, nachdem Sie die hartcodierten Konstanten ersetzt haben.

```
import json
import boto3
from datetime import datetime

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

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

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

            # Publish to CloudWatch
            response = cloudwatch.put_metric_data(
                Namespace='IoTSiteWise/AssetMetrics',
                MetricData=[
                    {
                        'MetricName': f'Property_your-property-id',
                        'Value': metric_value,
                        'Timestamp': timestamp,
                        'Dimensions': [
                            {
```

```
                            'Name': 'AssetId',
                            'Value': 'your-asset-id'
                        },
                        {
                            'Name': 'Quality',
                            'Value': quality
                        }
                    ]
                }
            ]
        )

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

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

Schritt 3: Erstellen AWS IoT Core Sie eine Regel für die Nachrichtenweiterleitung

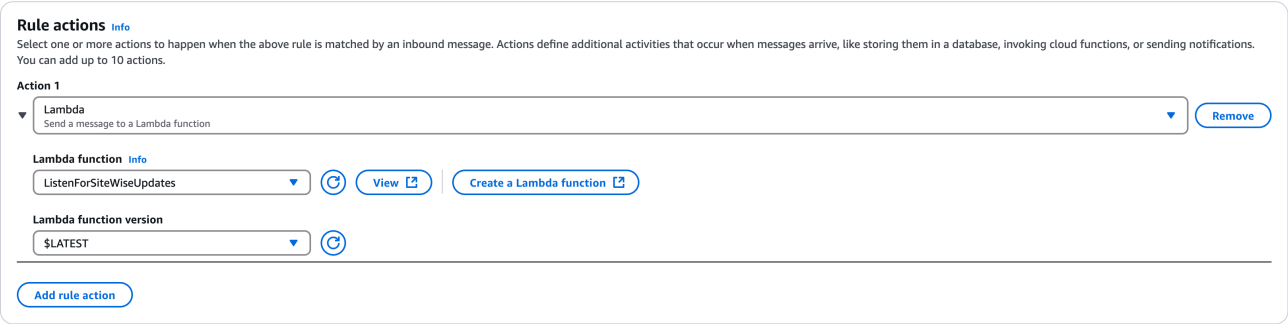
- Folgen Sie der Anleitung: Vorgang zum erneuten Veröffentlichen einer MQTT-Nachricht und geben Sie die folgenden Informationen ein, wenn Sie dazu aufgefordert werden:

  a. Nennen Sie die Nachrichten-Routing-Regel. `SiteWiseToCloudwatchAlarms`

  b. Für die Abfrage können Sie Folgendes verwenden:

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

  c. Wählen Sie unter Regelaktionen die Lambda-Aktion aus, an die die generierten Daten gesendet werden sollen AWS IoT SiteWise . CloudWatch Beispiel:



Schritt 4: Metriken anzeigen CloudWatch

Wenn Sie Daten in die zuvor ausgewählte Eigenschaft aufnehmen AWS IoT SiteWise, werden Daten an die Lambda-Funktion weitergeleitetSchritt 1: Aktivieren Sie MQTT-Benachrichtigungen auf der Asset-Eigenschaft, die wir in erstellt haben. Schritt 2: Erstellen Sie eine Funktion AWS Lambda In diesem Schritt können Sie überprüfen, an welches Lambda Ihre Metriken sendet. CloudWatch

1. Öffnen Sie das CloudWatch AWS-Managementkonsole.
2. Wählen Sie in der linken Navigationsleiste Metriken und dann Alle Metriken aus.
3. Wählen Sie die URL eines Alarms aus, um ihn zu öffnen.
4. Auf der Registerkarte Quelle sieht die CloudWatch Ausgabe ähnlich wie in diesem Beispiel aus. Diese Quellinformationen bestätigen, dass die metrischen Daten in CloudWatch.

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

Schritt 5: CloudWatch Alarme erstellen

Folgen Sie dem Verfahren Einen CloudWatch Alarm auf der Grundlage eines statischen Schwellenwerts erstellen im CloudWatch Amazon-Benutzerhandbuch, um Alarme für jede relevante Metrik zu erstellen.

> **Note**
>
> Es gibt viele Optionen für die Alarmkonfiguration in Amazon CloudWatch . Weitere Informationen zu CloudWatch Alarmen finden Sie unter Verwenden von CloudWatch Amazon-Alarmen im CloudWatch Amazon-Benutzerhandbuch.

Schritt 6: (Optional) importieren Sie den CloudWatch Alarm in AWS IoT SiteWise

Sie können CloudWatch Alarme so konfigurieren, dass Daten AWS IoT SiteWise mithilfe von CloudWatch Alarmaktionen und Lambda zurückgesendet werden. Diese Integration ermöglicht es Ihnen, Alarmstatus und -eigenschaften im SiteWise Monitor-Portal einzusehen.

1. Konfigurieren Sie den externen Alarm als Eigenschaft in einem Asset-Modell. Weitere Informationen finden Sie AWS IoT SiteWise im AWS IoT SiteWise Benutzerhandbuch unter Definieren externer Alarme.
2. Erstellen Sie eine Lambda-Funktion, die die BatchPutAssetPropertyValueAPI im AWS IoT SiteWise Benutzerhandbuch verwendet, um Alarmdaten an zu AWS IoT SiteWise senden.
3. Richten Sie CloudWatch Alarmaktionen ein, um Ihre Lambda-Funktion aufzurufen, wenn sich der Alarmstatus ändert. Weitere Informationen finden Sie im Abschnitt Alarmaktionen im CloudWatch Amazon-Benutzerhandbuch.

Einrichten AWS IoT Events

Dieser Abschnitt enthält eine Anleitung zur Einrichtung AWS IoT Events, einschließlich der Erstellung eines AWS Kontos, der Konfiguration der erforderlichen Berechtigungen und der Einrichtung von Rollen für die Verwaltung des Zugriffs auf Ressourcen.

Themen

Einrichtung eines AWS-Konto

Melde dich an für ein AWS-Konto

Wenn Sie noch keine haben AWS-Konto, führen Sie die folgenden Schritte aus, um eine zu erstellen.

Um sich für eine anzumelden AWS-Konto

1. Öffnen Sie https://portal.aws.amazon.com/billing/die Anmeldung.
2. Folgen Sie den Online-Anweisungen.

   Während der Anmeldung erhalten Sie einen Telefonanruf oder eine Textnachricht und müssen einen Verifizierungscode über die Telefontasten eingeben.

   Wenn Sie sich für eine anmelden AWS-Konto, Root-Benutzer des AWS-Kontoswird eine erstellt. Der Root-Benutzer hat Zugriff auf alle AWS-Services und Ressourcen des Kontos. Als bewährte Sicherheitsmethode weisen Sie einem Benutzer Administratorzugriff zu und verwenden Sie nur den Root-Benutzer, um Aufgaben auszuführen, die Root-Benutzerzugriff erfordern.

AWS sendet Ihnen nach Abschluss des Anmeldevorgangs eine Bestätigungs-E-Mail. Du kannst jederzeit deine aktuellen Kontoaktivitäten einsehen und dein Konto verwalten, indem du zu https://aws.amazon.com/gehst und Mein Konto auswählst.

Erstellen eines Benutzers mit Administratorzugriff

Nachdem Sie sich für einen angemeldet haben AWS-Konto, sichern Sie Ihren Root-Benutzer des AWS-Kontos AWS IAM Identity Center, aktivieren und erstellen Sie einen Administratorbenutzer, sodass Sie den Root-Benutzer nicht für alltägliche Aufgaben verwenden.

Sichern Sie Ihre Root-Benutzer des AWS-Kontos

1. Melden Sie sich AWS-Managementkonsoleals Kontoinhaber an, indem Sie Root-Benutzer auswählen und Ihre AWS-Konto E-Mail-Adresse eingeben. Geben Sie auf der nächsten Seite Ihr Passwort ein.

   Hilfe bei der Anmeldung mit dem Root-Benutzer finden Sie unter Anmelden als Root-Benutzer im AWS-Anmeldung -Benutzerhandbuch zu.
2. Aktivieren Sie die Multi-Faktor-Authentifizierung (MFA) für den Root-Benutzer.

   Anweisungen finden Sie unter Aktivieren eines virtuellen MFA-Geräts für Ihren AWS-Konto Root-Benutzer (Konsole) im IAM-Benutzerhandbuch.

Erstellen eines Benutzers mit Administratorzugriff

1. Aktivieren Sie das IAM Identity Center.

   Anweisungen finden Sie unter Aktivieren AWS IAM Identity Center im AWS IAM Identity Center -Benutzerhandbuch.
2. Gewähren Sie einem Administratorbenutzer im IAM Identity Center Benutzerzugriff.

   Ein Tutorial zur Verwendung von IAM-Identity-Center-Verzeichnis als Identitätsquelle finden Sie IAM-Identity-Center-Verzeichnis im Benutzerhandbuch unter Benutzerzugriff mit der Standardeinstellung konfigurieren.AWS IAM Identity Center

Anmelden als Administratorbenutzer

- Um sich mit Ihrem IAM-Identity-Center-Benutzer anzumelden, verwenden Sie die Anmelde-URL, die an Ihre E-Mail-Adresse gesendet wurde, als Sie den IAM-Identity-Center-Benutzer erstellt haben.

  Hilfe bei der Anmeldung mit einem IAM Identity Center-Benutzer finden Sie im AWS-Anmeldung Benutzerhandbuch unter Anmeldung beim AWS Access-Portal.

Weiteren Benutzern Zugriff zuweisen

1. Erstellen Sie im IAM-Identity-Center einen Berechtigungssatz, der den bewährten Vorgehensweisen für die Anwendung von geringsten Berechtigungen folgt.

   Anweisungen hierzu finden Sie unter Berechtigungssatz erstellen im AWS IAM Identity Center - Benutzerhandbuch.
2. Weisen Sie Benutzer einer Gruppe zu und weisen Sie der Gruppe dann Single Sign-On-Zugriff zu.

   Eine genaue Anleitung finden Sie unter Gruppen hinzufügen im AWS IAM Identity Center - Benutzerhandbuch.

Berechtigungen einrichten für AWS IoT Events

Die Implementierung geeigneter Berechtigungen ist wichtig für eine sichere und effektive Nutzung von AWS IoT Events. In diesem Abschnitt werden die Berechtigungen beschrieben, die für die Nutzung einiger Funktionen von erforderlich sind AWS IoT Events. Sie können AWS CLI Befehle oder die AWS Identity and Access Management (IAM-) Konsole verwenden, um Rollen und zugehörige Berechtigungsrichtlinien für den Zugriff auf Ressourcen oder die Ausführung bestimmter Funktionen in AWS IoT Events zu erstellen.

Das IAM-Benutzerhandbuch enthält detailliertere Informationen zur sicheren Steuerung von Zugriffsberechtigungen auf AWS Ressourcen. Spezifische Informationen zu AWS IoT Events finden Sie unter Aktionen, Ressourcen und Bedingungsschlüssel für AWS IoT Events.

Informationen zur Verwendung der IAM-Konsole zum Erstellen und Verwalten von Rollen und Berechtigungen finden Sie im IAM-Tutorial: AWS Kontoübergreifendes Delegieren des Zugriffs mithilfe von IAM-Rollen.

> **Note**
>
> Schlüssel können zwischen 1 und 128 Zeichen lang sein und Folgendes beinhalten:
>
> - Groß- oder Kleinbuchstaben a-z
> - Zahlen 0-9
> - Sonderzeichen -, _ oder:.

Aktionsberechtigungen für AWS IoT Events

AWS IoT Events ermöglicht es Ihnen, Aktionen auszulösen, die andere AWS Dienste verwenden. Dazu müssen Sie die AWS IoT Events Erlaubnis erteilen, diese Aktionen in Ihrem Namen durchzuführen. Dieser Abschnitt enthält eine Liste der Aktionen und eine Beispielrichtlinie, die die Erlaubnis erteilt, all diese Aktionen mit Ihren Ressourcen durchzuführen. Ändern Sie die *account-id* Verweise *region* und nach Bedarf. Wenn möglich, sollten Sie auch die Platzhalter (*) so ändern, dass sie sich auf bestimmte Ressourcen beziehen, auf die zugegriffen wird. Sie können die IAM-Konsole verwenden, um die Erlaubnis AWS IoT Events zum Senden einer von Ihnen definierten Amazon SNS SNS-Warnung zu erteilen.

AWS IoT Events unterstützt die folgenden Aktionen, mit denen Sie einen Timer verwenden oder eine Variable festlegen können:

- `setTimer`um einen Timer zu erstellen.
- `resetTimer`um den Timer zurückzusetzen.
- `clearTimer`um den Timer zu löschen.
- `setVariable`um eine Variable zu erstellen.

AWS IoT Events unterstützt die folgenden Aktionen, mit denen Sie mit AWS Diensten arbeiten können:

- `iotTopicPublish`um eine Nachricht zu einem MQTT-Thema zu veröffentlichen.
- `iotEvents`um Daten AWS IoT Events als Eingabewert an zu senden.
- `iotSiteWise` zum Senden von Daten an eine Komponenteneigenschaft in AWS IoT SiteWise
- `dynamoDB`um Daten an eine Amazon DynamoDB-Tabelle zu senden.
- `dynamoDBv2`um Daten an eine Amazon DynamoDB-Tabelle zu senden.
- `firehose`um Daten an einen Amazon Data Firehose-Stream zu senden.
- `lambda`um eine Funktion aufzurufen. AWS Lambda
- `sns`um Daten als Push-Benachrichtigung zu senden.
- `sqs`um Daten an eine Amazon SQS SQS-Warteschlange zu senden.

Example Richtlinie

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iot:Publish",
            "Resource": "arn:aws:iot:us-east-1:123456789012:topic/*"
        },
        {
            "Effect": "Allow",
            "Action": "iotevents:BatchPutMessage",
            "Resource": "arn:aws:iotevents:us-east-1:123456789012:input/*"
        },
        {
            "Effect": "Allow",
            "Action": "iotsitewise:BatchPutAssetPropertyValue",
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": "dynamodb:PutItem",
            "Resource": "arn:aws:dynamodb:us-east-1:123456789012:table/*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "firehose:PutRecord",
                "firehose:PutRecordBatch"
            ],
            "Resource": "arn:aws:firehose:us-east-1:123456789012:deliverystream/
*"
        },
        {
            "Effect": "Allow",
            "Action": "lambda:InvokeFunction",
            "Resource": "arn:aws:lambda:us-east-1:123456789012:function:*"
        },
        {
            "Effect": "Allow",
```

```
            "Action": "sns:Publish",
            "Resource": "arn:aws:sns:us-east-1:123456789012:*"
        },
        {
            "Effect": "Allow",
            "Action": "sqs:SendMessage",
            "Resource": "arn:aws:sqs:us-east-1:123456789012:*"
        }
    ]
}
```

Sicherung der Eingabedaten in AWS IoT Events

Es ist wichtig zu berücksichtigen, wer Zugriff auf Eingabedaten für die Verwendung in einem Detektormodell gewähren kann. Wenn Sie einen Benutzer oder eine Entität haben, deren Gesamtberechtigungen Sie einschränken möchten, die jedoch berechtigt sind, ein Detektormodell zu erstellen oder zu aktualisieren, müssen Sie diesem Benutzer oder dieser Entität auch die Erlaubnis erteilen, das Eingangs-Routing zu aktualisieren. Das bedeutet, dass Sie nicht nur die Erlaubnis für `iotevents:CreateDetectorModel` und erteilen müssen`iotevents:UpdateDetectorModel`, sondern auch die Erlaubnis für`iotevents:UpdateInputRouting`.

Example

Mit der folgenden Richtlinie wird die Erlaubnis für hinzugefügt`iotevents:UpdateInputRouting`.

JSON

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

Sie können anstelle des Platzhalters "*" eine Liste mit eingegebenen Amazon-Ressourcennamen (ARNs) angeben, um diese Berechtigung auf bestimmte Eingaben zu beschränken. `Resource` Auf diese Weise können Sie den Zugriff auf die Eingabedaten einschränken, die von Detektormodellen verwendet werden, die vom Benutzer oder der Entität erstellt oder aktualisiert wurden.

Amazon-Rollenrichtlinie für die CloudWatch Protokollierung für AWS IoT Events

Die folgenden Richtliniendokumente enthalten die Rollen- und Vertrauensrichtlinien, anhand AWS IoT Events derer Sie Protokolle in CloudWatch Ihrem Namen einreichen können.

Rollenrichtlinie:

JSON

```
{
    "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:*:*:*"
            ]
        }
    ]
}
```

Vertrauensrichtlinie:

JSON

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

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

Sie benötigen außerdem eine dem Benutzer zugeordnete IAM-Berechtigungsrichtlinie, die es dem Benutzer ermöglicht, Rollen zu übergeben, und zwar wie folgt. Weitere Informationen finden Sie im IAM-Benutzerhandbuch unter Gewähren von Benutzerberechtigungen zur AWS Übergabe einer Rolle an einen Dienst.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "",
            "Effect": "Allow",
            "Action": [
                "iam:GetRole",
                "iam:PassRole"
            ],
            "Resource": "arn:aws:iam::123456789012:role/Role_To_Pass"
        }
    ]
}
```

Sie können den folgenden Befehl verwenden, um die Ressourcenrichtlinie für CloudWatch Protokolle festzulegen. Dies ermöglicht AWS IoT Events das Einfügen von Protokollereignissen in CloudWatch Streams.

```
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\": \"*
\" } ] }"
```

Verwenden Sie den folgenden Befehl, um Protokollierungsoptionen festzulegen. Ersetzen Sie das `roleArn` durch die Protokollierungsrolle, die Sie erstellt haben.

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

Amazon SNS SNS-Messaging-Rollenrichtlinie für AWS IoT Events

Die Integration AWS IoT Events mit Amazon SNS erfordert ein sorgfältiges Berechtigungsmanagement für eine sichere und effiziente Zustellung von Benachrichtigungen. Dieser Leitfaden führt Sie durch den Prozess der Konfiguration von IAM-Rollen und -Richtlinien, um Nachrichten AWS IoT Events zu Amazon SNS SNS-Themen zu veröffentlichen.

Die folgenden Richtliniendokumente enthalten die Rollen- und Vertrauensrichtlinien, die das Senden von SNS-Nachrichten AWS IoT Events ermöglichen.

Rollenrichtlinie:

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Action": [
                "sns:*"
            ],
```

```
            "Effect": "Allow",
            "Resource": "arn:aws:sns:us-east-1:123456789012:testAction"
        }
    ]
}
```

Vertrauensrichtlinie:

JSON

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

Erste Schritte mit der AWS IoT Events Konsole

In diesem Abschnitt erfahren Sie, wie Sie mit der AWS IoT Events Konsole ein Eingabe- und ein Detektormodell erstellen. Sie modellieren zwei Zustände eines Motors: einen Normalzustand und einen Überdruckzustand. Wenn der gemessene Druck im Motor einen bestimmten Schwellenwert überschreitet, geht das Modell vom Normalzustand in den Überdruckzustand über. Dann sendet es eine Amazon SNS SNS-Nachricht, um einen Techniker über den Zustand zu informieren. Wenn der Druck bei drei aufeinanderfolgenden Druckmessungen wieder unter den Schwellenwert fällt, kehrt das Modell in den Normalzustand zurück und sendet eine weitere Amazon SNS SNS-Nachricht als Bestätigung.

Wir prüfen, ob drei aufeinanderfolgende Messwerte unter dem Druckschwellenwert liegen, um ein mögliches Stottern bei Überdruck oder normalen Meldungen im Falle einer nichtlinearen Erholungsphase oder einer anomalen Druckmessung zu vermeiden.

Auf der Konsole finden Sie auch mehrere vorgefertigte Modellvorlagen für Melder, die Sie anpassen können. Sie können die Konsole auch verwenden, um Meldermodelle zu importieren, die von anderen geschrieben wurden, oder um Ihre Meldermodelle zu exportieren und sie in verschiedenen AWS Regionen zu verwenden. Wenn Sie ein Meldermodell importieren, stellen Sie sicher, dass Sie die erforderlichen Eingaben erstellen oder sie für die neue Region neu erstellen, und aktualisieren Sie alle ARNs verwendeten Rollen.

Verwenden Sie die AWS IoT Events Konsole, um mehr über Folgendes zu erfahren.

Definieren Sie Eingaben

Um Ihre Geräte und Prozesse zu überwachen, müssen sie über eine Möglichkeit verfügen, Telemetriedaten in AWS IoT Events zu übertragen. Dies geschieht, indem Nachrichten als Eingaben an gesendet AWS IoT Events werden. Hierfür gibt es mehrere Möglichkeiten:

- Benutze die BatchPutMessageOperation.
- Schreiben Sie in AWS IoT Core eine AWS IoT Events Aktionsregel für die AWS IoT Regel-Engine, die Ihre Nachrichtendaten weiterleitet. AWS IoT Events Sie müssen die Eingabe anhand des Namens identifizieren.
- Verwenden Sie in AWS IoT Analytics die CreateDatasetOperation, um einen Datensatz mit zu erstellen`contentDeliveryRules`. Diese Regeln spezifizieren die AWS IoT Events Eingabe, an die der Inhalt des Datensatzes automatisch gesendet wird.

Bevor Ihre Geräte Daten auf diese Weise senden können, müssen Sie einen oder mehrere Eingänge definieren. Geben Sie dazu jeder Eingabe einen Namen und geben Sie an, welche Felder in den eingehenden Nachrichtendaten von der Eingabe überwacht werden.

Erstellen Sie ein Detektormodell

Definieren Sie mithilfe von Zuständen ein Detektormodell (ein Modell Ihrer Ausrüstung oder Ihres Prozesses). Definieren Sie für jeden Status eine bedingte (boolesche) Logik, die die eingehenden Eingaben auswertet, um signifikante Ereignisse zu erkennen. Wenn das Detektormodell ein Ereignis erkennt, kann es den Status ändern oder mithilfe anderer Dienste benutzerdefinierte oder vordefinierte Aktionen auslösen. AWS Sie können zusätzliche Ereignisse definieren, die Aktionen auslösen, wenn Sie einen Status betreten oder verlassen und optional, wenn eine Bedingung erfüllt ist.

In diesem Tutorial senden Sie eine Amazon SNS SNS-Nachricht als Aktion, wenn das Modell in einen bestimmten Status eintritt oder diesen verlässt.

Überwachen Sie ein Gerät oder einen Prozess

Wenn Sie mehrere Geräte oder Prozesse überwachen, geben Sie in jeder Eingabe ein Feld an, das das jeweilige Gerät oder den Prozess identifiziert, von dem die Eingabe stammt. Das `key` Feld finden Sie in`CreateDetectorModel`. Wenn das von identifizierte Eingabefeld einen neuen Wert `key` erkennt, wird ein neues Gerät identifiziert und ein Detektor erstellt. Jeder Detektor ist ein Exemplar des Detektormodells. Der neue Detektor reagiert weiterhin auf Eingaben von diesem Gerät, bis sein Detektormodell aktualisiert oder gelöscht wird.

Wenn Sie einen einzelnen Prozess überwachen (auch wenn mehrere Geräte oder Unterprozesse Eingaben senden), geben Sie kein eindeutiges `key` Identifikationsfeld an. In diesem Fall erzeugt das Modell einen einzelnen Detektor (Instanz), wenn die erste Eingabe eingeht.

Senden Sie Nachrichten als Eingaben an Ihr Detektormodell

Es gibt mehrere Möglichkeiten, eine Nachricht von einem Gerät oder Prozess als Eingabe in einen AWS IoT Events Detektor zu senden, ohne dass Sie die Nachricht zusätzlich formatieren müssen. In diesem Tutorial verwenden Sie die AWS IoT Konsole, um eine AWS IoT Events Aktionsregel für die AWS IoT Regel-Engine zu schreiben, in AWS IoT Events die Ihre Nachrichtendaten weitergeleitet werden.

Identifizieren Sie dazu die Eingabe anhand des Namens und verwenden Sie weiterhin die AWS IoT Konsole, um Nachrichten zu generieren, an die sie als Eingaben weitergeleitet AWS IoT Events werden.

> **Note**
>
> In diesem Tutorial wird die Konsole verwendet, um dasselbe `input` zu erstellen. Dies `detector model` wird im Beispiel unter gezeigtTutorials für AWS IoT Events Anwendungsfälle. Sie können dieses JSON-Beispiel verwenden, um dem Tutorial zu folgen.

Topics

Voraussetzungen für den Einstieg AWS IoT Events

Wenn Sie noch kein AWS Konto haben, erstellen Sie eines.

1. Folgen Sie den Anweisungen inEinrichten AWS IoT Events, um sicherzustellen, dass das Konto ordnungsgemäß eingerichtet und die erforderlichen Berechtigungen erteilt wurden.
2. Erstellen Sie zwei Amazon Simple Notification Service (Amazon SNS) -Themen.

   In diesem Tutorial (und dem entsprechenden Beispiel) wird davon ausgegangen, dass Sie zwei Amazon SNS SNS-Themen erstellt haben. Die ARNs dieser Themen werden wie folgt angezeigt: `arn:aws:sns:us-east-1:123456789012:underPressureAction` und`arn:aws:sns:us-east-1:123456789012:pressureClearedAction`. Ersetzen Sie diese Werte durch die ARNs von Ihnen Amazon SNS SNS-Themen. Weitere Informationen finden Sie im Amazon Simple Notification Service-Entwicklerhandbuch.

   Als Alternative zur Veröffentlichung von Benachrichtigungen zu Amazon SNS SNS-Themen können Sie die Detektoren MQTT-Nachrichten mit einem von Ihnen angegebenen Thema senden lassen. Mit dieser Option können Sie überprüfen, ob Ihr Detektormodell Instances erstellt und ob diese Instances Alerts senden, indem Sie die AWS IoT Core-Konsole verwenden, um Nachrichten zu abonnieren und zu überwachen, die zu diesen MQTT-Themen gesendet werden. Sie können den Namen des MQTT-Themas auch dynamisch zur Laufzeit definieren, indem Sie eine Eingabe oder Variable verwenden, die im Detektormodell erstellt wurde.

3. Wählen Sie einen AWS-Region , der unterstützt AWS IoT Events. Weitere Informationen finden Sie unter AWS IoT Events im Allgemeine AWS-Referenz. Hilfe finden Sie unter Erste Schritte mit einem Dienst im AWS-Managementkonsole im Abschnitt Erste Schritte mit dem AWS-Managementkonsole.

Erstellen Sie eine Eingabe für Modelle in AWS IoT Events

Wir empfehlen, bei der Erstellung der Eingaben für Ihre Modelle Dateien zusammenzustellen, die Beispielnachrichten enthalten, die Ihre Geräte oder Prozesse zur Meldung ihres Zustands senden. Diese Dateien helfen Ihnen dabei, die erforderlichen Eingaben zu definieren.

Sie können eine Eingabe mit mehreren Methoden erstellen, die in diesem Abschnitt beschrieben werden.

Erstellen Sie eine JSON-Eingabedatei

1. Erstellen Sie zunächst eine Datei mit dem Namen `input.json` auf Ihrem lokalen Dateisystem mit dem folgenden Inhalt:

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

2. Nachdem Sie diese `input.json` Starterdatei haben, können Sie eine Eingabe erstellen. Es gibt zwei Möglichkeiten, eine Eingabe zu erstellen. Sie können eine Eingabe mithilfe des Navigationsbereichs in der AWS IoT Events Konsole erstellen. Sie können auch eine Eingabe innerhalb des Detektormodells erstellen, nachdem es erstellt wurde.

Erstellen und konfigurieren Sie eine Eingabe

Erfahren Sie, wie Sie eine Eingabe für ein Alarmmodell oder ein Meldermodell erstellen.

1. Loggen Sie sich in die AWS IoT Events Konsole ein oder wählen Sie die Option Neues AWS IoT Events Konto erstellen.

2. Wählen Sie in der AWS IoT Events Konsole in der oberen linken Ecke den Navigationsbereich aus und erweitern Sie ihn.
3. Wählen Sie im linken Navigationsbereich Eingaben aus.
4. Wählen Sie in der rechten Ecke der Konsole die Option Eingabe erstellen aus.
5. Geben Sie ein Unikat an InputName.
6. Optional — geben Sie eine Beschreibung für Ihre Eingabe ein.
7. Um eine JSON-Datei hochzuladen, wählen Sie die `input.json` Datei aus, die Sie in der Übersicht für erstellt habenErstellen Sie eine JSON-Eingabedatei. Die Option Eingabeattribute auswählen wird mit einer Liste der von Ihnen eingegebenen Attribute angezeigt.
8. Wählen Sie unter Eingabeattribute auswählen die zu verwendenden Attribute aus und klicken Sie auf Erstellen. In diesem Beispiel wählen wir motorid und sensorData.Pressure aus.
9. Optional — Fügen Sie der Eingabe relevante Tags hinzu.

> **Note**
>
> Sie können auch zusätzliche Eingaben innerhalb des Meldermodells in der AWS IoT Events Konsole erstellen. Weitere Informationen finden Sie unter Erstellen Sie eine Eingabe innerhalb des Detektormodells in AWS IoT Events.

Erstellen Sie eine Eingabe innerhalb des Detektormodells in AWS IoT Events

Die Detektoreingänge AWS IoT Events dienen als Brücke zwischen Ihren Datenquellen und Detektormodellen. Die Detektoreingänge liefern die Rohdaten, die die Ereigniserkennungs- und Automatisierungsfunktionen von unterstützen AWS IoT Events. Erfahren Sie, wie Sie Detektoreingänge konfigurieren, damit Ihre Modelle präzise auf reale Ereignisse und Bedingungen in Ihrem IoT-Ökosystem reagieren können.

In diesem Abschnitt wird gezeigt, wie Sie einen Eingang für ein Detektormodell definieren, um Telemetriedaten oder Nachrichten zu empfangen.

Um eine Eingabe für ein Detektormodell zu definieren

1. Öffnen Sie die AWS IoT Events -Konsole.
2. Wählen Sie in der AWS IoT Events Konsole die Option Detektormodell erstellen aus.

3. Wählen Sie Neu erstellen.
4. Wählen Sie Create input (Eingabe erstellen).
5. Geben Sie für die Eingabe eine InputNameoptionale Beschreibung ein und wählen Sie Datei hochladen. Wählen Sie im daraufhin angezeigten Dialogfeld die `input.json` Datei aus, für die Sie in der Übersicht erstellt habenErstellen Sie eine JSON-Eingabedatei.
6. Wählen Sie unter Eingabeattribute auswählen die zu verwendenden Attribute aus und klicken Sie auf Erstellen. In diesem Beispiel wählen wir MotorID und SensorData.Pressure aus.

Erstellen Sie ein Detektormodell in AWS IoT Events

In diesem Thema definieren Sie mithilfe von Zuständen ein Detektormodell (ein Modell Ihrer Ausrüstung oder Ihres Prozesses).

Für jeden Status definieren Sie eine bedingte (boolesche) Logik, die die eingehenden Eingaben auswertet, um ein signifikantes Ereignis zu erkennen. Wenn ein Ereignis erkannt wird, ändert es den Status und kann zusätzliche Aktionen einleiten. Diese Ereignisse werden als Übergangsereignisse bezeichnet.

In Ihren Zuständen definieren Sie auch Ereignisse, die Aktionen ausführen können, wenn der Melder in diesen Zustand eintritt oder ihn verlässt oder wenn eine Eingabe eingeht (diese Ereignisse werden als `OnEnter` `OnInput` AND-Ereignisse bezeichnet). `OnExit` Die Aktionen werden nur ausgeführt, wenn die Bedingungslogik des Ereignisses Folgendes ergibt. `true`

So erstellen Sie ein Detektormodell

1. Der erste Melderstatus wurde für Sie erstellt. Um ihn zu ändern, wählen Sie den Kreis mit der Bezeichnung State_1 im Hauptbearbeitungsbereich aus.
2. Geben Sie im Bereich „Status“ den Namen des Bundesstaates ein und wählen Sie „OnEnterEreignis hinzufügen“.
3. Geben Sie auf der Seite „ OnEnter Ereignis hinzufügen“ einen Namen für das Ereignis und die Bedingung für das Ereignis ein. Geben Sie in diesem Beispiel ein, `true` um anzugeben, dass das Ereignis immer ausgelöst wird, wenn der Status eingegeben wird.
4. Wählen Sie unter Ereignisaktionen die Option Aktion hinzufügen aus.
5. Gehen Sie unter Ereignisaktionen wie folgt vor:

    a. Wählen Sie „Variable festlegen“

b. Wählen Sie für den Variablenbetrieb die Option Wert zuweisen aus.

c. Geben Sie unter Variablenname den Namen der Variablen ein, die festgelegt werden soll.

d. Geben Sie für Variablenwert den Wert **`0`** (Null) ein.

6. Wählen Sie Speichern.

   Eine Variable, wie die, die Sie definiert haben, kann auf jeden Fall im Detektormodell festgelegt werden (mit einem Wert). Auf den Wert der Variablen kann erst verwiesen werden (z. B. in der bedingten Logik eines Ereignisses), wenn der Detektor einen Status erreicht hat und eine Aktion ausgeführt hat, in der er definiert oder gesetzt ist.

7. Wählen Sie im Statusbereich das X neben Status, um zur Modellpalette Detector zurückzukehren.

8. Um einen zweiten Detektorstatus zu erstellen, wählen Sie in der Modellpalette Detektor die Option Status und ziehen Sie ihn in den Hauptbearbeitungsbereich. Dadurch wird ein Zustand mit dem Titel erstelltuntitled_state_1.

9. Halten Sie im ersten Status (Normal) an. Am Rand des Bundesstaats erscheint ein Pfeil.

10. Klicken Sie auf den Pfeil und ziehen Sie ihn vom ersten Status in den zweiten Status. Eine gerichtete Linie vom ersten zum zweiten Status (mit der Bezeichnung Unbenannt) wird angezeigt.

11. Wählen Sie die Linie Ohne Titel aus. Geben Sie im Bereich „Übergangsereignis“ einen Namen für das Ereignis und eine Logik für das Ereignis ein.

12. Wählen Sie im Bereich „Übergangsereignis“ die Option Aktion hinzufügen aus.

13. Wählen Sie im Bereich „Aktionen für Übergangsereignis hinzufügen“ die Option Aktion hinzufügen aus.

14. Wählen Sie für Aktion auswählen die Option Variable festlegen aus.

    a. Wählen Sie für Variablenoperation die Option Wert zuweisen aus.

    b. Geben Sie unter Variablenname den Namen der Variablen ein.

    c. Geben Sie für Wert zuweisen einen Wert ein, z. B.:
    `$variable.pressureThresholdBreached + 3`

    d. Wählen Sie Speichern.

15. Wählen Sie den zweiten Status untitled_state_1 aus.

16. Geben Sie im Bereich „Status“ den Namen des Bundesstaates ein und wählen Sie für „Bei Eingabe“ die Option „Ereignis hinzufügen“ aus.

17. Geben Sie auf der Seite „ OnEnter Ereignis hinzufügen“ den Namen des Ereignisses und die Bedingung für das Ereignis ein. Wählen Sie Aktion hinzufügen aus.
18. Wählen Sie für Aktion auswählen die Option SNS-Nachricht senden aus.
    a. Geben Sie unter SNS-Thema den Ziel-ARN Ihres Amazon SNS SNS-Themas ein.
    b. Wählen Sie Speichern.
19. Fahren Sie mit dem Hinzufügen der Ereignisse im Beispiel fort.
    a. Wählen Sie für OnInputEreignis hinzufügen und geben Sie die folgenden Ereignisinformationen ein und speichern Sie sie.

```
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. Wählen Sie für OnInput„Ereignis hinzufügen“ und geben Sie die folgenden Ereignisinformationen ein und speichern Sie sie.

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

    c. Wählen Sie für OnExitEreignis hinzufügen und geben Sie die folgenden Ereignisinformationen ein und speichern Sie sie unter Verwendung des ARN des Amazon SNS SNS-Themas, das Sie erstellt haben.

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

20. Halten Sie im zweiten Status (Gefährlich) an. Am Rand des Bundesstaats erscheint ein Pfeil
21. Klicken Sie auf den Pfeil und ziehen Sie ihn vom zweiten Status in den ersten Status. Eine gerichtete Linie mit der Bezeichnung Unbenannt wird angezeigt.
22. Wählen Sie die Zeile „Unbenannt“ und geben Sie im Bereich „Übergangsereignis“ anhand der folgenden Informationen einen Namen für das Ereignis und eine Logik für das Ereignis ein.

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

    Weitere Informationen darüber, warum wir den `$input` Wert und den `$variable` Wert in der Triggerlogik testen, finden Sie im Eintrag zur Verfügbarkeit von Variablenwerten unter. AWS IoT Events Einschränkungen und Einschränkungen des Detektormodells
23. Wählen Sie den Startstatus aus. Standardmäßig wurde dieser Status erstellt, als Sie ein Detektormodell erstellt haben). Wählen Sie im Startbereich den Zielstatus aus (z. B. Normal).
24. Als Nächstes konfigurieren Sie Ihr Meldermodell so, dass es auf Eingaben wartet. Wählen Sie in der oberen rechten Ecke Veröffentlichen.
25. Gehen Sie auf der Seite Meldermodell veröffentlichen wie folgt vor.

    a. Geben Sie einen Modellnamen für den Detektor, eine Beschreibung und den Namen einer Rolle ein. Diese Rolle wurde für Sie erstellt.

    b. Wählen Sie für jeden eindeutigen Schlüsselwert einen Detektor erstellen aus. Um Ihre eigene Rolle zu erstellen und zu verwenden, folgen Sie den Schritten unter Berechtigungen einrichten für AWS IoT Events und geben Sie sie hier als Rolle ein.
26. Wählen Sie unter Detector creation key den Namen eines der Attribute der Eingabe, die Sie zuvor definiert haben. Das Attribut, das Sie als Schlüssel für die Erstellung des Melders wählen, muss in jeder Nachrichteneingabe vorhanden sein und für jedes Gerät, das Nachrichten sendet, eindeutig sein. In diesem Beispiel wird das motorid-Attribut verwendet.
27. Wählen Sie Save and publish (Speichern und veröffentlichen).

> **Note**
>
> Die Anzahl der eindeutigen Melder, die für ein bestimmtes Detektormodell erstellt wurden, basiert auf den gesendeten Eingangsmeldungen. Wenn ein Detektormodell erstellt wird,

wird ein Schlüssel aus den Eingabeattributen ausgewählt. Dieser Schlüssel bestimmt, welche Detektorinstanz verwendet werden soll. Wenn der Schlüssel noch nie gesehen wurde (für dieses Detektormodell), wird eine neue Detektorinstanz erstellt. Wenn der Schlüssel schon einmal gesehen wurde, verwenden wir die bestehende Detektorinstanz, die diesem Schlüsselwert entspricht.

Sie können eine Sicherungskopie Ihrer Detektormodelldefinition (in JSON) erstellen, das Detektormodell neu erstellen oder aktualisieren oder als Vorlage verwenden, um ein anderes Detektormodell zu erstellen.

Sie können dies von der Konsole aus oder mit dem folgenden CLI-Befehl tun. Ändern Sie bei Bedarf den Namen des Detektormodells so, dass er dem entspricht, den Sie bei der Veröffentlichung im vorherigen Schritt verwendet haben.

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

Dadurch wird eine Datei (`motorDetectorModel.json`) erstellt, deren Inhalt dem folgenden ähnelt.

```
{
    "detectorModel": {
        "detectorModelConfiguration": {
            "status": "ACTIVE",
            "lastUpdateTime": 1552072424.212,
            "roleArn": "arn:aws:iam::123456789012:role/IoTEventsRole",
            "creationTime": 1552072424.212,
            "detectorModelArn": "arn:aws:iotevents:us-
west-2:123456789012:detectorModel/motorDetectorModel",
            "key": "motorid",
            "detectorModelName": "motorDetectorModel",
            "detectorModelVersion": "1"
        },
        "detectorModelDefinition": {
            "states": [
                {
                    "onInput": {
                        "transitionEvents": [
                            {
                                "eventName": "Overpressurized",
                                "actions": [
```

```
                                    {
                                        "setVariable": {
                                            "variableName":
"pressureThresholdBreached",
                                            "value":
"$variable.pressureThresholdBreached + 3"
                                        }
                                    }
                                ],
                                "condition": "$input.PressureInput.sensorData.pressure
> 70",
                                "nextState": "Dangerous"
                            }
                        ],
                        "events": []
                    },
                    "stateName": "Normal",
                    "onEnter": {
                        "events": [
                            {
                                "eventName": "init",
                                "actions": [
                                    {
                                        "setVariable": {
                                            "variableName":
"pressureThresholdBreached",
                                            "value": "0"
                                        }
                                    }
                                ],
                                "condition": "true"
                            }
                        ]
                    },
                    "onExit": {
                        "events": []
                    }
                },
                {
                    "onInput": {
                        "transitionEvents": [
                            {
                                "eventName": "Back to Normal",
                                "actions": [],
```

```
                                "condition": "$variable.pressureThresholdBreached <= 1
&& $input.PressureInput.sensorData.pressure <= 70",
                                "nextState": "Normal"
                            }
                        ],
                        "events": [
                            {
                                "eventName": "Overpressurized",
                                "actions": [
                                    {
                                        "setVariable": {
                                            "variableName":
"pressureThresholdBreached",
                                            "value": "3"
                                        }
                                    }
                                ],
                                "condition": "$input.PressureInput.sensorData.pressure
> 70"
                            },
                            {
                                "eventName": "Pressure Okay",
                                "actions": [
                                    {
                                        "setVariable": {
                                            "variableName":
"pressureThresholdBreached",
                                            "value":
"$variable.pressureThresholdBreached - 1"
                                        }
                                    }
                                ],
                                "condition": "$input.PressureInput.sensorData.pressure
<= 70"
                            }
                        ]
                    },
                    "stateName": "Dangerous",
                    "onEnter": {
                        "events": [
                            {
                                "eventName": "Pressure Threshold Breached",
                                "actions": [
                                    {
```

```
                                        "sns": {
                                            "targetArn": "arn:aws:sns:us-
west-2:123456789012:MyIoTButtonSNSTopic"
                                        }
                                    }
                                ],
                                "condition": "$variable.pressureThresholdBreached > 1"
                            }
                        ]
                    },
                    "onExit": {
                        "events": [
                            {
                                "eventName": "Normal Pressure Restored",
                                "actions": [
                                    {
                                        "sns": {
                                            "targetArn": "arn:aws:sns:us-
west-2:123456789012:IoTVirtualButtonTopic"
                                        }
                                    }
                                ],
                                "condition": "true"
                            }
                        ]
                    }
                }
            ],
            "initialStateName": "Normal"
        }
    }
}
```

Senden Sie Eingaben zum Testen des Detektormodells in AWS IoT Events

Es gibt mehrere Möglichkeiten, Telemetriedaten zu empfangen AWS IoT Events (siehe Unterstützte Aktionen zum Empfangen von Daten und Auslösen von Aktionen in AWS IoT Events). In diesem Thema erfahren Sie, wie Sie in der AWS IoT Konsole eine AWS IoT Regel erstellen, die Nachrichten als Eingaben an Ihren AWS IoT Events Detektor weiterleitet. Sie können den MQTT-Client der AWS IoT Konsole verwenden, um Testnachrichten zu senden. Sie können diese Methode verwenden, um

Telemetriedaten darüber abzurufen, AWS IoT Events wann Ihre Geräte MQTT-Nachrichten über den AWS IoT Message Broker senden können.

Um Eingaben zu senden, um das Detektormodell zu testen

1. Öffnen Sie die AWS IoT Core -Konsole. Wählen Sie im linken Navigationsbereich unter Verwalten die Option Nachrichtenweiterleitung und anschließend Regeln aus.
2. Wählen Sie oben rechts Regel erstellen aus.
3. Führen Sie auf der Seite Regel erstellen die folgenden Schritte aus:

   1. Schritt 1. Geben Sie die Eigenschaften der Regel an. Füllen Sie die folgenden Felder aus:
      - Name der Regel. Geben Sie einen Namen für Ihre Regel ein, z. `MyIoTEventsRule` B.

        > **Note**
        >
        > Verwenden Sie keine Leerzeichen.

      - Beschreibung der Regel. Das ist optional.
      - Wählen Sie Weiter aus.
   2. Schritt 2. Konfigurieren Sie die SQL-Anweisung. Füllen Sie die folgenden Felder aus:
      - SQL-Version. Wählen Sie die entsprechende Option aus der Liste aus.
      - SQL-Anweisung. Geben Sie **`SELECT *, topic(2) as motorid FROM 'motors/+/status'`** ein.

      Wählen Sie Weiter aus.
   3. Schritt 3. Regelaktionen anhängen Gehen Sie im Abschnitt Regelaktionen wie folgt vor:
      - Aktion 1. Wählen Sie IoT Events aus. Die folgenden Felder werden angezeigt:
        a. Geben Sie den Namen ein. Wählen Sie die entsprechende Option aus der Liste aus. Wenn Ihre Eingabe nicht angezeigt wird, wählen Sie Aktualisieren.

        Um eine neue Eingabe zu erstellen, wählen Sie Create IoT Events input aus. Füllen Sie die folgenden Felder aus:

        - Geben Sie den Namen ein. Geben Sie `PressureInput` ein.
        - Beschreibung. Das ist optional.

- Laden Sie eine JSON-Datei hoch. Laden Sie eine Kopie Ihrer JSON-Datei hoch. Auf diesem Bildschirm befindet sich ein Link zu einer Beispieldatei, falls Sie keine Datei haben. Der Code beinhaltet:

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

- Wählen Sie Eingabeattribute. Wählen Sie die entsprechende (n) Option (en) aus.
- Tags. Dies ist optional.

Wählen Sie Erstellen aus.

Kehren Sie zum Bildschirm Regel erstellen zurück und aktualisieren Sie das Feld Eingabename. Wählen Sie die Eingabe aus, die Sie gerade erstellt haben.

b. Batch-Modus. Das ist optional. Wenn es sich bei der Payload um ein Array von Nachrichten handelt, wählen Sie diese Option.

c. Nachrichten-ID. Dies ist zwar optional, wird aber empfohlen.

d. IAM role (IAM-Rolle. Wählen Sie die entsprechende Rolle aus der Liste aus. Wenn die Rolle nicht aufgeführt ist, wählen Sie Neue Rolle erstellen aus.

Geben Sie einen Rollennamen ein und wählen Sie Erstellen aus.

Um eine weitere Regel hinzuzufügen, wählen Sie Regelaktion hinzufügen

- Fehleraktion. Dieser Abschnitt ist optional. Um eine Aktion hinzuzufügen, wählen Sie Fehleraktion hinzufügen und wählen Sie die entsprechende Aktion aus der Liste aus.

Füllen Sie die angezeigten Felder aus.

- Wählen Sie Weiter aus.

4. Schritt 4. Überprüfen und erstellen. Überprüfen Sie die Informationen auf dem Bildschirm und wählen Sie Erstellen.

4. Wählen Sie im linken Navigationsbereich unter Test die Option MQTT-Testclient aus.

5. Wählen Sie Publish to topic (In Thema veröffentlichen) aus. Füllen Sie die folgenden Felder aus:

- Name des Themas. Geben Sie einen Namen ein, um die Nachricht zu identifizieren, z. `motors/Fulton-A32/status` B.
- Nutzlast der Nachricht. Geben Sie Folgendes ein:

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

> **Note**
>
> Ändern Sie die `messageId` jedes Mal, wenn Sie eine neue Nachricht veröffentlichen.

6. Behalten Sie für Publish das gleiche Thema bei, ändern Sie das `"pressure"` in der Payload jedoch auf einen Wert, der über dem Schwellenwert liegt, den Sie im Detektormodell angegeben haben (z. B.**85**).
7. Wählen Sie Publish.

Die von Ihnen erstellte Detector-Instance generiert und sendet Ihnen eine Amazon SNS SNS-Nachricht. Senden Sie weiterhin Nachrichten mit Druckwerten über oder unter dem Druckgrenzwert (70 in diesem Beispiel), um zu sehen, wie der Detektor in Betrieb ist.

In diesem Beispiel müssen Sie drei Nachrichten mit Druckwerten unter dem Schwellenwert senden, um in den Normalzustand zurückzukehren und eine Amazon SNS SNS-Meldung zu erhalten, die darauf hinweist, dass der Überdruckzustand behoben ist. Sobald der Melder wieder im Normalzustand ist, wechselt der Detektor durch eine Meldung mit einem Druckwert über dem Grenzwert in den Status Gefährlich und sendet eine Amazon SNS SNS-Meldung, die auf diesen Zustand hinweist.

Nachdem Sie nun ein einfaches Eingabe- und Meldermodell erstellt haben, versuchen Sie Folgendes.

- Weitere Beispiele (Vorlagen) für Detektormodelle finden Sie auf der Konsole.
- Folgen Sie den Schritten unterErstellen Sie mit CLI einen AWS IoT Events Detektor für zwei Zustände, um ein Eingabe- und Detektormodell mit dem zu erstellen AWS CLI

- Erfahren Sie mehr über die in den Ereignissen Ausdrücke zum Filtern, Transformieren und Verarbeiten von Ereignisdaten verwendeten.
- Erfahren Sie mehr über Unterstützte Aktionen zum Empfangen von Daten und Auslösen von Aktionen in AWS IoT Events.
- Wenn etwas nicht funktioniert, finden Sie weitere Informationen unterProblembehebung AWS IoT Events.

Bewährte Verfahren für AWS IoT Events

Folgen Sie diesen bewährten Methoden, um den größtmöglichen Nutzen daraus zu ziehen AWS IoT Events.

Topics

Aktivieren Sie die CloudWatch Amazon-Protokollierung bei der Entwicklung von AWS IoT Events Detektormodellen

Amazon CloudWatch überwacht Ihre AWS Ressourcen und die Anwendungen, auf denen Sie laufen, AWS in Echtzeit. Damit CloudWatch erhalten Sie systemweiten Einblick in die Ressourcennutzung, die Anwendungsleistung und den Betriebszustand. Wenn Sie ein AWS IoT Events Detektormodell entwickeln oder debuggen, CloudWatch wissen Sie, was AWS IoT Events gerade passiert und welche Fehler dabei auftreten.

Um zu aktivieren CloudWatch

1. Falls Sie dies noch nicht getan haben, folgen Sie den Schritten unter Berechtigungen einrichten für AWS IoT Events So erstellen Sie eine Rolle mit einer angehängten Richtlinie, die die Berechtigung zum Erstellen und Verwalten von CloudWatch Protokollen für erteilt AWS IoT Events.
2. Rufen Sie die AWS IoT Events -Konsole auf.
3. Wählen Sie im Navigationsbereich Settings (Einstellungen).
4. Wählen Sie auf der Seite Einstellungen die Option Bearbeiten aus.
5. Gehen Sie auf der Seite Protokollierungsoptionen bearbeiten im Abschnitt Protokollierungsoptionen wie folgt vor:

    a. Wählen Sie unter Ausführlichkeitsgrad eine Option aus.

    b. Wählen Sie unter Rolle auswählen eine Rolle aus, die über ausreichende Berechtigungen verfügt, um die von Ihnen ausgewählten Protokollierungsaktionen durchzuführen.

c. (Optional) Wenn Sie Debug als Ausführlichkeitsstufe ausgewählt haben, können Sie Debug-Ziele hinzufügen, indem Sie wie folgt vorgehen:

   i. Wählen Sie unter Debug-Ziele die Option Modelloption hinzufügen aus.

   ii. Geben Sie einen Modellnamen für den Detektor ein und (optional) KeyValue, um die Meldermodelle und spezifischen Melder (Instanzen) anzugeben, die protokolliert werden sollen.

6. Wählen Sie Aktualisieren aus.

Ihre Protokollierungsoptionen wurden erfolgreich aktualisiert.

Veröffentlichen Sie regelmäßig, um Ihr Meldermodell zu speichern, wenn Sie in der AWS IoT Events Konsole arbeiten

Wenn Sie die AWS IoT Events Konsole verwenden, werden Ihre laufenden Arbeiten lokal in Ihrem Browser gespeichert. Sie müssen jedoch Veröffentlichen wählen, um Ihr Meldermodell zu speichern AWS IoT Events. Nachdem Sie ein Detektormodell veröffentlicht haben, ist Ihr veröffentlichtes Werk in jedem Browser verfügbar, den Sie für den Zugriff auf Ihr Konto verwenden.

> **Note**
>
> Wenn Sie Ihr Werk nicht veröffentlichen, wird es nicht gespeichert. Nachdem Sie ein Detektormodell veröffentlicht haben, können Sie seinen Namen nicht mehr ändern. Sie können seine Definition jedoch weiter ändern.

Tutorials für AWS IoT Events Anwendungsfälle

AWS IoT Events Tutorials bieten eine Sammlung von Verfahren, die verschiedene Aspekte abdecken AWS IoT Events, von der Grundkonfiguration bis hin zu spezielleren Anwendungsfällen. Jedes Tutorial zeigt Beispiele für praktische Szenarien und hilft Ihnen dabei, praktische Fähigkeiten bei der Erstellung von Detektormodellen, der Konfiguration von Eingängen, der Einrichtung von Aktionen und der Integration mit anderen AWS Diensten zu erwerben, um leistungsstarke IoT-Lösungen zu erstellen.

In diesem Kapitel erfahren Sie, wie Sie:

- Holen Sie sich Hilfe bei der Entscheidung, welche Zustände in Ihr Detektormodell aufgenommen werden sollen, und bestimmen Sie, ob Sie eine oder mehrere Detektorinstanzen benötigen.
- Folgen Sie einem Beispiel, das den verwendet AWS CLI.
- Erstellen Sie einen Eingang für den Empfang von Telemetriedaten von einem Gerät und einem Detektormodell, um den Status des Geräts, das diese Daten sendet, zu überwachen und darüber zu berichten.
- Informieren Sie sich über die Einschränkungen und Beschränkungen für Eingänge, Meldermodelle und den AWS IoT Events Service.
- Sehen Sie sich ein komplexeres Beispiel für ein Detektormodell mit Kommentaren an.

Topics

AWS IoT Events Zur Überwachung Ihrer IoT-Geräte verwenden

Sie können AWS IoT Events damit Ihre Geräte oder Prozesse überwachen und bei wichtigen Ereignissen Maßnahmen ergreifen. Gehen Sie dazu wie folgt vor:

Eingaben erstellen

Sie müssen über eine Möglichkeit verfügen, mit der Ihre Geräte und Prozesse Telemetriedaten abrufen können. AWS IoT Events Sie tun dies, indem Sie Nachrichten als Eingaben an AWS IoT Events senden. Sie können Nachrichten auf verschiedene Arten als Eingaben senden:

- Verwenden Sie die BatchPutMessageOperation.
- Definieren Sie eine `iotEvents`Regelaktion für die AWS IoT Core Regel-Engine. Die Regelaktion leitet Nachrichtendaten aus Ihrer Eingabe an. AWS IoT Events
- Verwenden Sie in die CreateDatasetOperation AWS IoT Analytics, um einen Datensatz mit zu erstellen. `contentDeliveryRules` Diese Regeln spezifizieren die AWS IoT Events Eingabe, an die der Inhalt des Datensatzes automatisch gesendet wird.
- Definieren Sie eine IoTEvents-Aktion in einem AWS IoT Events Detektormodell `onExit` oder `transitionEvents` in einem `onInput` Ereignis. Informationen über die Detektormodellinstanz und das Ereignis, das die Aktion ausgelöst hat, werden als Eingabe mit dem von Ihnen angegebenen Namen an das System zurückgemeldet.

Bevor Ihre Geräte Daten auf diese Weise senden, müssen Sie einen oder mehrere Eingänge definieren. Geben Sie dazu jeder Eingabe einen Namen und geben Sie an, welche Felder in den eingehenden Nachrichtendaten von der Eingabe überwacht werden. AWS IoT Events erhält seine Eingabe in Form von JSON-Nutzdaten aus vielen Quellen. Jede Eingabe kann eigenständig bearbeitet oder mit anderen Eingaben kombiniert werden, um komplexere Ereignisse zu erkennen.

Erstellen Sie ein Detektormodell

Definieren Sie mithilfe von Zuständen ein Detektormodell (ein Modell Ihrer Ausrüstung oder Ihres Prozesses). Für jeden Zustand definieren Sie eine bedingte (boolesche) Logik, die die eingehenden Eingaben auswertet, um signifikante Ereignisse zu erkennen. Wenn ein Ereignis erkannt wird, kann es den Status ändern oder mithilfe anderer AWS Dienste benutzerdefinierte oder vordefinierte Aktionen auslösen. Sie können zusätzliche Ereignisse definieren, die Aktionen auslösen, wenn Sie einen Status betreten oder verlassen und optional, wenn eine Bedingung erfüllt ist.

In diesem Tutorial senden Sie eine Amazon SNS SNS-Nachricht als Aktion, wenn das Modell in einen bestimmten Status eintritt oder diesen verlässt.

Überwachen Sie ein Gerät oder einen Prozess

Wenn Sie mehrere Geräte oder Prozesse überwachen, geben Sie in jeder Eingabe ein Feld an, das das jeweilige Gerät oder den Prozess identifiziert, von dem die Eingabe stammt. (Siehe das

`key` Feld unter`CreateDetectorModel`.) Wenn ein neues Gerät identifiziert wird (ein neuer Wert wird in dem Eingabefeld angezeigt, das durch den identifiziert wird`key`), wird ein Detektor erzeugt. (Jeder Detektor ist eine Instanz des Detektormodells.) Dann reagiert der neue Detektor weiterhin auf Eingaben von diesem Gerät, bis sein Meldermodell aktualisiert oder gelöscht wird.

Wenn Sie einen einzelnen Prozess überwachen (auch wenn mehrere Geräte oder Unterprozesse Eingaben senden), geben Sie kein eindeutiges `key` Identifikationsfeld an. In diesem Fall wird ein einzelner Detektor (Instanz) erstellt, wenn die erste Eingabe eintrifft.

Senden Sie Nachrichten als Eingaben an Ihr Detektormodell

Es gibt mehrere Möglichkeiten, eine Nachricht von einem Gerät oder Prozess als Eingabe in einen AWS IoT Events Detektor zu senden, ohne dass Sie die Nachricht zusätzlich formatieren müssen. In diesem Tutorial verwenden Sie die AWS IoT Konsole, um eine AWS IoT Events Aktionsregel für die AWS IoT Core Regel-Engine zu schreiben, in AWS IoT Events die Ihre Nachrichtendaten weitergeleitet werden. Dazu identifizieren Sie die Eingabe anhand des Namens. Anschließend verwenden Sie weiterhin die AWS IoT Konsole, um einige Nachrichten zu generieren, an die Sie als Eingaben weitergeleitet werden AWS IoT Events.

Woher wissen Sie, welche Zustände Sie in einem Detektormodell benötigen?

Um zu bestimmen, welche Zustände Ihr Detektormodell haben sollte, entscheiden Sie zunächst, welche Maßnahmen Sie ergreifen können. Wenn Ihr Auto beispielsweise mit Benzin betrieben wird, schauen Sie bei Fahrtantritt auf die Tankanzeige, ob Sie tanken müssen. Hier haben Sie eine Aktion: Sagen Sie dem Fahrer, er solle „Gas holen“. Ihr Meldermodell benötigt zwei Zustände: „Auto benötigt keinen Kraftstoff“ und „Auto benötigt Kraftstoff“. Im Allgemeinen möchten Sie einen Status für jede mögliche Aktion und einen weiteren für den Fall definieren, dass keine Aktion erforderlich ist. Das funktioniert auch dann, wenn die Aktion selbst komplizierter ist. Möglicherweise möchten Sie Informationen darüber suchen, wo sich die nächstgelegene Tankstelle oder der günstigste Preis befindet, und diese Informationen hinzufügen, aber Sie tun dies, wenn Sie die Nachricht „Geh tanken“ senden.

Um zu entscheiden, welchen Status Sie als Nächstes eingeben möchten, schauen Sie sich die Eingaben an. Die Eingaben enthalten die Informationen, die Sie benötigen, um zu entscheiden, in welchem Zustand Sie sich befinden sollten. Um eine Eingabe zu erstellen, wählen Sie eines oder mehrere Felder in einer von Ihrem Gerät oder Prozess gesendeten Nachricht aus, die Ihnen bei der Entscheidung helfen. In diesem Beispiel benötigen Sie eine Eingabe, die Ihnen den aktuellen

Kraftstoffstand („Prozent voll“) anzeigt. Vielleicht sendet Ihnen Ihr Auto mehrere verschiedene Nachrichten mit jeweils unterschiedlichen Feldern. Um diese Eingabe zu erstellen, müssen Sie die Nachricht und das Feld auswählen, das den aktuellen Füllstand der Gasanzeige anzeigt. Die Länge der Reise, die Sie unternehmen werden („Entfernung zum Ziel“), kann aus Gründen der Übersichtlichkeit fest codiert werden. Sie können Ihre durchschnittliche Reisedauer verwenden. Auf der Grundlage der Eingabe führen Sie einige Berechnungen durch (wie viele Gallonen entspricht dieser volle Prozentsatz? ist die durchschnittliche Reisedauer größer als die Meilen, die Sie zurücklegen können, wenn man die Gallonen berücksichtigt, die Sie haben, und Ihren durchschnittlichen „Meilen pro Gallone“. Sie führen diese Berechnungen durch und senden bei Ereignissen Nachrichten.

Bisher haben Sie zwei Zustände und eine Eingabe. Sie benötigen ein Ereignis im ersten Status, das die Berechnungen auf der Grundlage der Eingabe durchführt und entscheidet, ob in den zweiten Status übergegangen werden soll. Das ist ein Übergangsereignis. (`transitionEvents`befinden sich in der `onInput` Ereignisliste eines Bundesstaates. Beim Empfang einer Eingabe in diesem ersten Zustand wechselt das Ereignis in den zweiten Status, sofern der Zustand des Ereignisses erfüllt `condition` ist.) Wenn Sie den zweiten Status erreichen, senden Sie die Nachricht, sobald Sie den Status betreten. (Sie verwenden ein `onEnter` Ereignis. Beim Eintritt in den zweiten Status sendet dieses Ereignis die Nachricht. Sie müssen nicht warten, bis eine weitere Eingabe eintrifft.) Es gibt auch andere Arten von Ereignissen, aber das ist alles, was Sie für ein einfaches Beispiel benötigen.

Die anderen Arten von Ereignissen sind `onExit` und`onInput`. Sobald eine Eingabe eingeht und die Bedingung erfüllt ist, führt ein `onInput` Ereignis die angegebenen Aktionen aus. Wenn ein Vorgang seinen aktuellen Status verlässt und die Bedingung erfüllt ist, führt das `onExit` Ereignis die angegebenen Aktionen aus.

Fehlt dir etwas? Ja, wie kehrt man zum ersten Zustand „Auto braucht keinen Kraftstoff“ zurück? Nachdem Sie Ihren Benzintank gefüllt haben, zeigt der Eingang an, dass der Tank voll ist. In Ihrem zweiten Zustand benötigen Sie ein Übergangsereignis, das zum ersten Zustand zurückkehrt. Dieses Ereignis tritt ein, wenn die Eingabe empfangen wird (in den `onInput:` Ereignissen des zweiten Zustands). Es sollte in den ersten Zustand zurückkehren, wenn die Berechnungen ergeben, dass Sie jetzt genug Benzin haben, um dorthin zu gelangen, wo Sie hin möchten.

Das sind die Grundlagen. Einige Detektormodelle werden komplexer, wenn sie Zustände hinzufügen, die wichtige Eingaben widerspiegeln, nicht nur mögliche Aktionen. In einem Detektormodell, das die Temperatur erfasst, könnten beispielsweise drei Zustände vorliegen: ein „normaler“ Zustand, ein „zu heißer“ Zustand und ein Zustand mit „potenziellem Problem“. Sie gehen in den potenziellen

Problemzustand über, wenn die Temperatur über ein bestimmtes Niveau steigt, aber noch nicht zu heiß geworden ist. Sie möchten keinen Alarm senden, es sei denn, die Temperatur bleibt länger als 15 Minuten bei dieser Temperatur. Wenn sich die Temperatur vorher wieder normalisiert hat, geht der Melder wieder in den Normalzustand über. Wenn der Timer abläuft, wechselt der Melder in den zu heißen Zustand und sendet einen Alarm, nur um vorsichtig zu sein. Sie könnten dasselbe tun, indem Sie Variablen und einen komplexeren Satz von Ereignisbedingungen verwenden. Oft ist es jedoch einfacher, einen anderen Status zu verwenden, um die Ergebnisse Ihrer Berechnungen tatsächlich zu speichern.

Woher wissen Sie, ob Sie eine oder mehrere Instanzen eines Detektors benötigen?

Um zu entscheiden, wie viele Instanzen Sie benötigen, fragen Sie sich: „Was möchten Sie wissen?“ Nehmen wir an, Sie möchten wissen, wie das Wetter heute ist. Regnet es (Bundesstaat)? Müssen Sie einen Regenschirm mitnehmen (Aktion)? Sie können einen Sensor verwenden, der die Temperatur meldet, einen anderen, der die Luftfeuchtigkeit meldet, und andere, die den Luftdruck, die Windgeschwindigkeit und -richtung sowie den Niederschlag melden. Sie müssen jedoch alle diese Sensoren gemeinsam überwachen, um den Wetterzustand (Regen, Schnee, Bewölkung, Sonne) und die entsprechenden Maßnahmen zu ermitteln (nehmen Sie sich einen Regenschirm oder tragen Sie Sonnencreme auf). Trotz der Anzahl der Sensoren möchten Sie, dass eine Melderinstanz den Wetterstatus überwacht und Sie darüber informiert, welche Maßnahmen zu ergreifen sind.

Wenn Sie jedoch für die Wettervorhersage in Ihrer Region verantwortlich sind, verfügen Sie möglicherweise über mehrere Instanzen solcher Sensoranordnungen, die sich an verschiedenen Orten in der Region befinden. Die Menschen an jedem Standort müssen wissen, wie das Wetter an diesem Ort ist. In diesem Fall benötigen Sie mehrere Instanzen Ihres Melders. Die von jedem Sensor an jedem Standort gemeldeten Daten müssen ein Feld enthalten, das Sie als `key` Feld festgelegt haben. Dieses Feld ermöglicht es, eine Melderinstanz für den Bereich AWS IoT Events zu erstellen und diese Informationen dann weiterhin an diese Melderinstanz weiterzuleiten, sobald sie eintreffen. Keine ruinierten Haare oder sonnenverbrannten Nasen mehr!

Im Grunde benötigen Sie eine Melderinstanz, wenn Sie eine Situation (einen Prozess oder einen Standort) überwachen müssen. Wenn Sie viele Situationen (Standorte, Prozesse) überwachen müssen, benötigen Sie mehrere Melderinstanzen.

Erstellen Sie mit CLI einen AWS IoT Events Detektor für zwei Zustände

In diesem Beispiel rufen wir die AWS CLI Befehle AWS IoT Events APIs using auf, um einen Detektor zu erstellen, der zwei Zustände eines Motors modelliert: einen Normalzustand und einen Überdruckzustand.

Wenn der gemessene Druck im Motor einen bestimmten Schwellenwert überschreitet, wechselt das Modell in den Überdruckzustand und sendet eine Amazon Simple Notification Service (Amazon SNS) -Meldung, um einen Techniker über den Zustand zu informieren. Wenn der Druck bei drei aufeinanderfolgenden Druckmessungen unter den Schwellenwert fällt, kehrt das Modell in den Normalzustand zurück und sendet eine weitere Amazon SNS SNS-Nachricht als Bestätigung, dass der Zustand behoben ist. Wir benötigen drei aufeinanderfolgende Messwerte unter dem Druckschwellenwert, um bei einer nichtlinearen Erholungsphase oder einem einmaligen anomalen Wiederaufnahmemeldungen ein Stottern der Meldungen zu vermeiden.

Im Folgenden finden Sie eine Übersicht über die Schritte zur Erstellung des Melders.

Erstellen Sie Eingaben.

Um Ihre Geräte und Prozesse zu überwachen, müssen sie über eine Möglichkeit verfügen, Telemetriedaten in AWS IoT Events zu übertragen. Dies geschieht, indem Nachrichten als Eingaben an gesendet AWS IoT Events werden. Hierfür gibt es mehrere Möglichkeiten:

- Benutze die BatchPutMessageOperation. Diese Methode ist einfach, setzt jedoch voraus, dass Ihre Geräte oder Prozesse über ein SDK oder das auf die AWS IoT Events API zugreifen können AWS CLI.
- Schreiben Sie in AWS IoT Core eine AWS IoT Events Aktionsregel für die AWS IoT Core Regel-Engine, in AWS IoT Events die Ihre Nachrichtendaten weitergeleitet werden. Dadurch wird die Eingabe anhand des Namens identifiziert. Verwenden Sie diese Methode, wenn Ihre Geräte oder Prozesse Nachrichten senden können oder dies bereits tun AWS IoT Core. Diese Methode erfordert in der Regel weniger Rechenleistung von einem Gerät.
- Verwenden Sie die CreateDatasetOperation AWS IoT Analytics, um einen Datensatz mit `contentDeliveryRules` Angabe der AWS IoT Events Eingabe zu erstellen, wobei der Inhalt des Datensatzes automatisch gesendet wird. Verwenden Sie diese Methode, wenn Sie Ihre Geräte oder Prozesse auf der Grundlage aggregierter oder analysierter Daten steuern möchten. AWS IoT Analytics

Bevor Ihre Geräte Daten auf diese Weise senden können, müssen Sie eine oder mehrere Eingaben definieren. Geben Sie dazu jeder Eingabe einen Namen und geben Sie an, welche Felder in den eingehenden Nachrichtendaten von der Eingabe überwacht werden sollen.

Erstellen Sie ein Detektormodell

Erstellen Sie mithilfe von Zuständen ein Detektormodell (ein Modell Ihrer Ausrüstung oder Ihres Prozesses). Definieren Sie für jeden Status eine bedingte (boolesche) Logik, die die eingehenden Eingaben auswertet, um signifikante Ereignisse zu erkennen. Wenn ein Ereignis erkannt wird, kann es den Status ändern oder mithilfe anderer Dienste benutzerdefinierte oder vordefinierte Aktionen auslösen. AWS Sie können zusätzliche Ereignisse definieren, die Aktionen auslösen, wenn Sie einen Status betreten oder verlassen und optional, wenn eine Bedingung erfüllt ist.

Überwachen Sie mehrere Geräte oder Prozesse

Wenn Sie mehrere Geräte oder Prozesse überwachen und diese einzeln verfolgen möchten, geben Sie in jeder Eingabe ein Feld an, das das jeweilige Gerät oder den Prozess identifiziert, von dem die Eingabe stammt. Sehen Sie sich das `key` Feld in an`CreateDetectorModel`. Wenn ein neues Gerät identifiziert wird (ein neuer Wert wird in dem durch das identifizierten Eingabefeld angezeigt`key`), wird eine Melderinstanz erstellt. Die neue Melderinstanz reagiert weiterhin auf Eingaben von diesem bestimmten Gerät, bis ihr Meldermodell aktualisiert oder gelöscht wird. Sie haben so viele eindeutige Detektoren (Instanzen), wie es eindeutige Werte in den `key` Eingabefeldern gibt.

Überwachen Sie ein einzelnes Gerät oder einen einzelnen Prozess

Wenn Sie einen einzelnen Prozess überwachen (auch wenn mehrere Geräte oder Unterprozesse Eingaben senden), geben Sie kein eindeutiges `key` Identifikationsfeld an. In diesem Fall wird ein einzelner Detektor (Instanz) erstellt, wenn die erste Eingabe eintrifft. Beispielsweise könnten Sie in jedem Raum eines Hauses Temperatursensoren haben, aber nur eine HLK-Einheit, um das gesamte Haus zu heizen oder zu kühlen. Sie können dies also nur als einen einzigen Vorgang steuern, auch wenn jeder Raumnutzer möchte, dass seine Stimme (Eingabe) Vorrang hat.

Senden Sie Nachrichten von Ihren Geräten oder Prozessen als Eingaben in Ihr Meldermodell

In den Eingängen haben wir die verschiedenen Möglichkeiten beschrieben, eine Nachricht von einem Gerät oder Prozess als Eingabe in einen AWS IoT Events Detektor zu senden. Nachdem Sie die Eingänge erstellt und das Detektormodell erstellt haben, können Sie mit dem Senden von Daten beginnen.

> **Note**
>
> Wenn Sie ein Meldermodell erstellen oder ein vorhandenes aktualisieren, dauert es einige Minuten, bis das neue oder aktualisierte Meldermodell Nachrichten empfängt und Melder (Instanzen) erstellt. Wenn das Meldermodell aktualisiert wird, kann es sein, dass Sie während dieser Zeit weiterhin ein Verhalten beobachten, das auf der vorherigen Version basiert.

Topics

Erstellen Sie einen AWS IoT Events Eingang zur Erfassung von Gerätedaten

Bei der Einrichtung von Eingängen für AWS IoT Events können Sie mithilfe der definieren AWS CLI , wie Ihre Geräte Sensordaten kommunizieren. Wenn Ihre Geräte beispielsweise Nachrichten im JSON-Format mit Motorkennungen und Sensormesswerten senden, können Sie diese Daten erfassen, indem Sie eine Eingabe erstellen, die den Nachrichten bestimmte Attribute wie den Druck und die Motor-ID zuordnet. Der Prozess beginnt mit der Definition einer Eingabe in einer JSON-Datei, der Angabe der relevanten Datenpunkte und der Verwendung von, AWS CLI um die Eingabe zu registrieren. AWS IoT Events Auf diese Weise können kritische Bedingungen AWS IoT auf der Grundlage von Sensordaten in Echtzeit überwacht und darauf reagiert werden.

Nehmen wir als Beispiel an, Ihre Geräte senden Nachrichten im folgenden Format.

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

Mit dem folgenden AWS CLI Befehl können Sie eine Eingabe erstellen, um die `pressure` Daten und die `motorid` (die das spezifische Gerät identifiziert, das die Nachricht gesendet hat) zu erfassen.

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

Die Datei `pressureInput.json` enthält Folgendes.

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

Wenn Sie Ihre eigenen Eingaben erstellen, denken Sie daran, zunächst Beispielnachrichten als JSON-Dateien von Ihren Geräten oder Prozessen zu sammeln. Sie können sie verwenden, um eine Eingabe über die Konsole oder die CLI zu erstellen.

Erstellen Sie ein Meldermodell zur Darstellung von Gerätezuständen in AWS IoT Events

In Erstellen Sie einen AWS IoT Events Eingang zur Erfassung von Gerätedaten haben Sie ein auf einer Nachricht `input` basierendes System erstellt, das Druckdaten von einem Motor meldet. Um mit dem Beispiel fortzufahren: Hier ist ein Detektormodell, das auf ein Überdruckereignis in einem Motor reagiert.

Sie erstellen zwei Zustände: "`Normal`„und"`Dangerous`“. Jeder Detektor (Instanz) geht bei seiner Erstellung in den Zustand `Normal` "" über. Die Instanz wird erstellt, wenn eine Eingabe mit einem eindeutigen Wert für `key` "`motorid`" eingeht.

Wenn die Melder-Instance einen Druckwert von 70 oder mehr empfängt, wechselt sie in den Status `Dangerous` "" und sendet eine Amazon SNS SNS-Nachricht als Warnung. Wenn die Druckwerte bei drei aufeinanderfolgenden Eingängen wieder normal sind (weniger als 70), kehrt der Detektor in den Zustand "`Normal`" zurück und sendet eine weitere Amazon SNS SNS-Meldung als Entwarnung.

Dieses Beispiel-Detektormodell geht davon aus, dass Sie zwei Amazon SNS SNS-Themen erstellt haben, deren Amazon-Ressourcennamen (ARNs) in der Definition als `"targetArn": "arn:aws:sns:us-east-1:123456789012:underPressureAction"` und `"targetArn": "arn:aws:sns:us-east-1:123456789012:pressureClearedAction"` angezeigt werden.

Weitere Informationen finden Sie im Amazon Simple Notification Service Developer Guide und insbesondere in der Dokumentation des CreateTopicVorgangs in der Amazon Simple Notification Service API-Referenz.

In diesem Beispiel wird auch davon ausgegangen, dass Sie eine AWS Identity and Access Management (IAM-) Rolle mit den entsprechenden Berechtigungen erstellt haben. Der ARN dieser Rolle wird in der Definition des Detektormodells als angezeigt`"roleArn": "arn:aws:iam::123456789012:role/IoTEventsRole"`. Folgen Sie den Schritten unterBerechtigungen einrichten für AWS IoT Events, um diese Rolle zu erstellen, und kopieren Sie den ARN der Rolle an die entsprechende Stelle in der Definition des Detektormodells.

Sie können das Detektormodell mit dem folgenden AWS CLI Befehl erstellen.

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

Die Datei `"motorDetectorModel.json"` enthält Folgendes.

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

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

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

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

Senden Sie Nachrichten als Eingaben an einen Detektor in AWS IoT Events

Sie haben jetzt eine Eingabe definiert, die die wichtigen Felder in Nachrichten identifiziert, die von einem Gerät gesendet werden (siehe Erstellen Sie einen AWS IoT Events Eingang zur Erfassung von Gerätedaten). Im vorherigen Abschnitt haben Sie eine erstellt, `detector model` die auf ein Überdruckereignis in einem Motor reagiert (siehe Erstellen Sie ein Meldermodell zur Darstellung von Gerätezuständen in AWS IoT Events).

Um das Beispiel zu vervollständigen, senden Sie Nachrichten von einem Gerät (in diesem Fall einem Computer, auf dem der Computer AWS CLI installiert ist) als Eingänge an den Melder.

> **Note**
>
> Wenn Sie ein Meldermodell erstellen oder ein vorhandenes aktualisieren, dauert es einige Minuten, bis das neue oder aktualisierte Meldermodell Nachrichten empfängt und Melder (Instanzen) erstellt. Wenn Sie das Meldermodell aktualisieren, kann es sein, dass Sie während dieser Zeit weiterhin ein Verhalten beobachten, das auf der vorherigen Version basiert.

Verwenden Sie den folgenden AWS CLI Befehl, um eine Nachricht mit Daten zu senden, die den Schwellenwert überschreiten.

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

Die Datei "`highPressureMessage.json`" enthält Folgendes.

```
{
  "messages": [
    {
      "messageId": "00001",
      "inputName": "PressureInput",
      "payload": "{\"motorid\": \"Fulton-A32\", \"sensorData\": {\"pressure\": 80,
 \"temperature\": 39} }"
    }
  ]
}
```

Sie müssen das `messageId` in jeder gesendeten Nachricht ändern. Wenn Sie es nicht ändern, dedupliziert das AWS IoT Events System die Nachrichten. AWS IoT Events ignoriert eine Nachricht, wenn sie dieselbe enthält `messageID` wie eine andere Nachricht, die innerhalb der letzten fünf Minuten gesendet wurde.

An diesem Punkt wird ein Detektor (Instanz) erstellt, um Ereignisse für den Motor `"Fulton-A32"` zu überwachen. Dieser Detektor geht in den `"Normal"` Zustand über, in dem er erstellt wurde. Da wir jedoch einen Druckwert über dem Schwellenwert gesendet haben, wechselt er sofort in den `"Dangerous"` Status. Dabei sendet der Detektor eine Nachricht an den Amazon SNS SNS-Endpunkt, dessen ARN lautet`arn:aws:sns:us-east-1:123456789012:underPressureAction`.

Führen Sie den folgenden AWS CLI Befehl aus, um eine Nachricht mit Daten zu senden, die unter dem Druckschwellenwert liegen.

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

Die Datei `normalPressureMessage.json` enthält Folgendes.

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

```
}
```

Sie müssen das `messageId` in der Datei jedes Mal ändern, wenn Sie den `BatchPutMessage` Befehl innerhalb von fünf Minuten aufrufen. Senden Sie die Nachricht noch zweimal. Nachdem die Nachricht dreimal gesendet wurde, sendet der Detektor (Instance) für den Motor "`Fulton-A32`" eine Nachricht an den Amazon SNS SNS-Endpunkt "`arn:aws:sns:us-east-1:123456789012:pressureClearedAction`" und wechselt erneut in den "`Normal`" Status.

> **Note**
>
> Sie können mehrere Nachrichten gleichzeitig mit senden. `BatchPutMessage` Die Reihenfolge, in der diese Nachrichten verarbeitet werden, kann jedoch nicht garantiert werden. Um sicherzustellen, dass Nachrichten (Eingaben) in der richtigen Reihenfolge verarbeitet werden, senden Sie sie nacheinander und warten Sie bei jedem API-Aufruf auf eine erfolgreiche Antwort.

Im Folgenden finden Sie Beispiele für Nutzdaten von SNS-Nachrichten, die mit dem in diesem Abschnitt beschriebenen Beispiel für das Detektormodell erstellt wurden.

bei Ereignis „Druckschwellenwert überschritten“

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

```
      },
      "timers":{}
    }
  },
  "eventName":"Pressure Threshold Breached"
}
```

bei Ereignis „Normaldruck wiederhergestellt“

```
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"
}
```

Wenn Sie Timer definiert haben, wird deren aktueller Status auch in den Payloads der SNS-Nachrichten angezeigt.

Die Nachrichtennutzdaten enthalten Informationen über den Status des Detektors (Instanz) zum Zeitpunkt des Sendens der Nachricht (d. h. zum Zeitpunkt der Ausführung der SNS-Aktion). Sie können den https://docs.aws.amazon.com/iotevents/latest/apireference/API_iotevents-data_DescribeDetector.html Vorgang verwenden, um ähnliche Informationen über den Status des Melders zu erhalten.

AWS IoT Events Einschränkungen und Einschränkungen des Detektormodells

Bei der Erstellung eines Detektormodells sind die folgenden Punkte zu beachten.

Wie benutzt man das **`actions`** Feld

Das `actions` Feld ist eine Liste von Objekten. Sie können mehr als ein Objekt haben, aber in jedem Objekt ist nur eine Aktion zulässig.

Example

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

Wie benutzt man das **`condition`** Feld

Das `condition` ist erforderlich für `transitionEvents` und in anderen Fällen optional.

Wenn das `condition` Feld nicht vorhanden ist, entspricht es`"condition": true`.

Das Ergebnis der Auswertung eines Bedingungsausdrucks sollte ein boolescher Wert sein. Wenn das Ergebnis kein boolescher Wert ist, entspricht es dem im Ereignis angegebenen Wert `false` und leitet den Übergang zu dem im `actions` Ereignis `nextState` angegebenen Wert nicht ein.

Verfügbarkeit von Variablenwerten

Wenn der Wert einer Variablen in einem Ereignis festgelegt wird, ist ihr neuer Wert standardmäßig nicht verfügbar oder wird nicht verwendet, um Bedingungen in anderen Ereignissen in derselben

Gruppe auszuwerten. Der neue Wert ist nicht verfügbar oder wird in einer Ereignisbedingung im gleichen `onInput` `onExit` Feld `onEnter` oder verwendet.

Stellen Sie den `evaluationMethod` Parameter in der Definition des Detektormodells ein, um dieses Verhalten zu ändern. Wenn der auf gesetzt `evaluationMethod` ist`SERIAL`, werden Variablen aktualisiert und die Ereignisbedingungen werden in der Reihenfolge ausgewertet, in der die Ereignisse definiert sind. Andernfalls werden Variablen innerhalb eines Zustands aktualisiert, und Ereignisse innerhalb eines Zustands werden erst ausgeführt, wenn alle Ereignisbedingungen ausgewertet wurden, wenn für `BATCH` oder standardmäßig dieser Wert festgelegt ist. `evaluationMethod`

Der `"Dangerous"` Status im `onInput` Feld `"$variable.pressureThresholdBreached"` wird um eins dekrementiert, wenn `"Pressure Okay"` die Bedingung erfüllt ist (wenn der aktuelle Eingangsdruck kleiner oder gleich 70 ist).

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

Der Detektor sollte wieder in den `"Normal"` Zustand zurückkehren, wenn 0 `"$variable.pressureThresholdBreached"` erreicht ist (d. h. wenn der Detektor drei aufeinanderfolgende Druckwerte erhalten hat, die kleiner oder gleich 70 sind). Das `"BackToNormal"` Ereignis `transitionEvents` muss testen, ob der Wert kleiner oder gleich 1 (nicht 0) `"$variable.pressureThresholdBreached"` ist. Außerdem muss erneut überprüft werden, ob der aktuelle Wert von kleiner oder gleich 70 `"$input.PressureInput.sensorData.pressure"` ist.

```
"transitionEvents": [
  {
```

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

Andernfalls, wenn die Bedingung nur den Wert der Variablen prüft, würden zwei normale Messwerte, gefolgt von einer Überdruckmessung, die Bedingung erfüllen und in den `"Normal"` Zustand zurückkehren. Bei der Bedingung wird der Wert berücksichtigt, der bei der vorherigen Verarbeitung einer Eingabe angegeben `"$variable.pressureThresholdBreached"` wurde. Der Wert der Variablen wird in diesem `"Overpressurized"` Fall auf 3 zurückgesetzt, aber denken Sie daran, dass dieser neue Wert noch für niemanden verfügbar ist`condition`.

Standardmäßig `condition` kann jedes Mal, wenn ein Steuerelement das `onInput` Feld betritt, der Wert einer Variablen nur so sehen, wie er zu Beginn der Verarbeitung der Eingabe war, bevor er durch die unter angegebenen Aktionen geändert wird`onInput`. Das Gleiche gilt für `onEnter` und`onExit`. Jede Änderung, die an einer Variablen vorgenommen wird, wenn wir den Status betreten oder verlassen, ist nicht für andere Bedingungen verfügbar, die in denselben `onEnter` oder `onExit` -Feldern angegeben sind.

Latenz bei der Aktualisierung eines Detektormodells

Wenn Sie ein Detektormodell aktualisieren, löschen und neu erstellen (siehe UpdateDetectorModel), kommt es zu einer gewissen Verzögerung, bis alle generierten Detektoren (Instanzen) gelöscht werden und das neue Modell zur Neuerstellung der Detektoren verwendet wird. Sie werden neu erstellt, nachdem das neue Detektormodell wirksam wird und neue Eingaben eintreffen. Während dieser Zeit werden Eingaben möglicherweise weiterhin von den Detektoren verarbeitet, die von der vorherigen Version des Detektormodells erzeugt wurden. Während dieses Zeitraums erhalten Sie möglicherweise weiterhin Warnmeldungen, die durch das vorherige Meldermodell definiert wurden.

Leerzeichen in den Eingabetasten

Leerzeichen sind in Eingabeschlüsseln zulässig, aber Verweise auf den Schlüssel müssen in Backticks eingeschlossen werden, und zwar sowohl in der Definition des Eingabeattributs als auch dann, wenn der Wert des Schlüssels in einem Ausdruck referenziert wird. Zum Beispiel bei einer Nachrichtennutzlast wie der folgenden:

```
{
```

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

Verwenden Sie Folgendes, um die Eingabe zu definieren.

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

In einem bedingten Ausdruck müssen Sie auch mithilfe von Backticks auf den Wert eines solchen Schlüssels verweisen.

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

Ein kommentiertes Beispiel: HVAC-Temperaturregelung mit AWS IoT Events

Einige der folgenden JSON-Beispieldateien enthalten eingebettete Kommentare, was sie zu ungültigen JSON-Dateien macht. Vollständige Versionen dieser Beispiele ohne Kommentare finden Sie unterBeispiel: Verwendung der HVAC-Temperatursteuerung mit AWS IoT Events.

In diesem Beispiel wird ein Thermostatsteuerungsmodell implementiert, das Ihnen folgende Möglichkeiten bietet.

- Definieren Sie nur ein Meldermodell, das zur Überwachung und Steuerung mehrerer Bereiche verwendet werden kann. Für jeden Bereich wird eine Melderinstanz erstellt.
- Erfassen Sie Temperaturdaten von mehreren Sensoren in jedem Kontrollbereich.

- Ändern Sie den Temperatursollwert für einen Bereich.
- Stellen Sie die Betriebsparameter für jeden Bereich ein und setzen Sie diese Parameter zurück, während die Instanz verwendet wird.
- Dynamisches Hinzufügen oder Löschen von Sensoren aus einem Bereich.
- Geben Sie eine Mindestlaufzeit an, um Heiz- und Kühlgeräte zu schützen.
- Abnormale Sensormesswerte zurückweisen.
- Definieren Sie Notfall-Sollwerte, die sofort Heizen oder Kühlen einschalten, wenn ein Sensor eine Temperatur über oder unter einem bestimmten Schwellenwert meldet.
- Melden Sie anomale Messwerte und Temperaturspitzen.

Themen

Eingabedefinitionen für Detektormodelle in AWS IoT Events

Wir möchten ein Detektormodell erstellen, mit dem wir die Temperatur in verschiedenen Bereichen überwachen und steuern können. Jeder Bereich kann über mehrere Sensoren verfügen, die die Temperatur melden. Wir gehen davon aus, dass jeder Bereich von einer Heizeinheit und einer Kühleinheit versorgt wird, die ein- oder ausgeschaltet werden können, um die Temperatur in dem Bereich zu regeln. Jeder Bereich wird von einer Melderinstanz gesteuert.

Da die verschiedenen Bereiche, die wir überwachen und steuern, unterschiedliche Eigenschaften aufweisen können, die unterschiedliche Steuerparameter erfordern, definieren wir die `'seedTemperatureInput'` so, dass diese Parameter für jeden Bereich bereitgestellt werden. Wenn wir eine dieser Eingangsnachrichten an senden AWS IoT Events, wird eine neue

Detektormodellinstanz erstellt, die die Parameter enthält, die wir in diesem Bereich verwenden möchten. Hier ist die Definition dieser Eingabe.

CLI-Befehl:

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

Datei: `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" }
    ]
  }
}
```

Antwort:

```
{
    "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."
    }
}
```

Hinweise

- Für jedes in einer Nachricht `'areaId'` empfangene Unikat wird eine neue Detektorinstanz erstellt. Sehen Sie sich das `'key'` Feld in der `'areaDetectorModel'` Definition an.
- Die Durchschnittstemperatur kann `'desiredTemperature'` von dem abweichen, `'allowedError'` bevor die Heiz- oder Kühlgeräte für den Bereich aktiviert werden.
- Meldet ein Sensor eine Temperatur über dem`'rangeHigh'`, meldet der Melder einen Temperaturanstieg und startet sofort die Kühleinheit.
- Meldet ein Sensor eine Temperatur unter dem`'rangeLow'`, meldet der Melder einen Temperaturanstieg und startet sofort die Heizeinheit.
- Wenn ein Sensor eine Temperatur über `'anomalousHigh'` oder unter dem Wert meldet`'anomalousLow'`, meldet der Melder einen anomalen Sensorwert, ignoriert jedoch den gemeldeten Temperaturwert.
- Das `'sensorCount'` teilt dem Melder mit, wie viele Sensoren für den Bereich Bericht erstatten. Der Detektor berechnet die Durchschnittstemperatur in dem Gebiet, indem er jedem Temperaturmesswert, den er empfängt, den entsprechenden Gewichtungsfaktor zuweist. Aus diesem Grund muss der Detektor nicht nachverfolgen, was jeder Sensor meldet, und die Anzahl der Sensoren kann je nach Bedarf dynamisch geändert werden. Wenn jedoch ein einzelner Sensor offline geht, weiß der Melder dies nicht und berücksichtigt es auch nicht. Wir empfehlen Ihnen, ein anderes Meldermodell speziell für die Überwachung des Verbindungsstatus der einzelnen Sensoren zu erstellen. Zwei sich ergänzende Detektormodelle vereinfachen das Design beider.
- Der `'noDelay'` Wert kann `true` oder sein`false`. Nach dem Einschalten eines Heiz- oder Kühlgeräts sollte es für eine bestimmte Mindestzeit eingeschaltet bleiben, um die Integrität des Geräts zu schützen und seine Lebensdauer zu verlängern. Wenn auf eingestellt `'noDelay'` ist`false`, erzwingt die Melderinstanz eine Verzögerung, bevor sie die Kühl- und Heizgeräte ausschaltet, um sicherzustellen, dass sie so lange wie möglich laufen. Die Anzahl der Sekunden der Verzögerung wurde in der Definition des Detektormodells fest codiert, da wir keinen Variablenwert verwenden können, um einen Timer einzustellen.

Der `'temperatureInput'` wird verwendet, um Sensordaten an eine Detektorinstanz zu übertragen.

CLI-Befehl:

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

Datei: `temperatureInput.json`

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

Antwort:

```
{
    "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."
    }
}
```

Hinweise

- Die wird `'sensorId'` nicht von einer Beispiel-Detektorinstanz verwendet, um einen Sensor direkt zu steuern oder zu überwachen. Es wird automatisch an Benachrichtigungen weitergegeben, die von der Detektorinstanz gesendet werden. Von dort aus kann es verwendet werden, um die Sensoren zu identifizieren, die ausfallen (z. B. könnte ein Sensor, der regelmäßig ungewöhnliche Messwerte sendet, bald ausfallen) oder die offline gegangen sind (wenn er als Eingang für ein zusätzliches Meldermodell verwendet wird, das den Herzschlag des Geräts überwacht). `'sensorId'`Sie können auch dabei helfen, warme oder kalte Zonen in einem Gebiet zu identifizieren, wenn die Messwerte regelmäßig vom Durchschnitt abweichen.
- Das `'areaId'` wird verwendet, um die Daten des Sensors an die entsprechende Melderinstanz weiterzuleiten. Für jedes in einer Nachricht `'areaId'` empfangene Unikat wird eine

Detektorinstanz erstellt. Sehen Sie sich das `'key'` Feld in der `'areaDetectorModel'` Definition an.

Erstellen Sie eine Definition für ein AWS IoT Events Detektormodell

Das `'areaDetectorModel'` Beispiel enthält Kommentare in der Zeile.

CLI-Befehl:

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

Datei: `areaDetectorModel.json`

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

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

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

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

```
            "actions": [
              {
                "sns": {
                  "targetArn": "arn:aws:sns:us-west-2:123456789012:heatOff"
                }
              },
              {
                "sns": {
                  "targetArn": "arn:aws:sns:us-west-2:123456789012:coolOff"
                }
              },
              {
                "iotTopicPublish": {
                  "mqttTopic": "hvac/Heating/Off"
                }
              },
              {
                "iotTopicPublish": {
                  "mqttTopic": "hvac/Cooling/Off"
                }
              }
            ]
          }
        ]
      }
    },


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

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

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

          {
            "eventName": "highTemperatureSpike",
            "condition": "$input.temperatureInput.sensorData.temperature >
$variable.rangeHigh",
            // When even a single temperature reading arrives that is above the
'rangeHigh', take
            //   emergency action to begin cooling, and report a high temperature
spike.
            "actions": [
              {
                "iotTopicPublish": {
                  "mqttTopic": "temperatureSensor/spike"
                }
              },
              {
                "sns": {
                  "targetArn": "arn:aws:sns:us-west-2:123456789012:coolOn"
                }
              },
```

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

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

```
              {
                "setVariable": {
                  "variableName": "enteringNewState",
                  "value": "true"
                }
              }
            ],
            "nextState": "heating"
          },

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

          {
```

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


    {
      "stateName": "cooling",
      "onEnter": {
        "events": [
          {
            "eventName": "delay",
            "condition": "!$variable.noDelay && $variable.enteringNewState",
```

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

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

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

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

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

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

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

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

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

```
        }
      },


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

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

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

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

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

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

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

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

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

  ],

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

Antwort:

```
{
```

```
    "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"
    }
}
```

Wird verwendet BatchUpdateDetector , um ein AWS IoT Events Meldermodell zu aktualisieren

Sie können den `BatchUpdateDetector` Vorgang verwenden, um eine Detektorinstanz in einen bekannten Zustand zu versetzen, einschließlich Timer- und Variablenwerten. Im folgenden Beispiel werden die `BatchUpdateDetector` Betriebsparameter für einen Bereich zurückgesetzt, der unter Temperaturüberwachung und -steuerung steht. Mit diesem Vorgang können Sie dies tun, ohne das Meldermodell löschen, neu erstellen oder aktualisieren zu müssen.

CLI-Befehl:

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

Datei: `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": [
        ]
      }
    }
  ]
}
```

Antwort:

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

Verwenden Sie BatchPutMessage für Eingaben in AWS IoT Events

Example 1

Verwenden Sie den `BatchPutMessage` Vorgang, um eine `"seedTemperatureInput"` Nachricht zu senden, in der die Betriebsparameter für einen bestimmten Bereich festgelegt werden, der temperaturgesteuert und überwacht wird. Jede Nachricht AWS IoT Events , die von dieser empfangen wird, `"areaId"` hat eine neue Melderinstanz zur Folge. Die neue Melderinstanz ändert ihren Status jedoch nicht `"idle"` und beginnt erst, die Temperatur zu überwachen und Heiz- oder Kühlgeräte zu steuern, wenn eine `"seedTemperatureInput"` Meldung für den neuen Bereich eingeht.

CLI-Befehl:

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

Datei: `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}"
    }
  ]
}
```

Antwort:

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

Example

2

Verwenden Sie den `BatchPutMessage` Vorgang, um eine `"temperatureInput"` Nachricht zu senden, um Temperatursensordaten für einen Sensor in einem bestimmten Steuerungs- und Überwachungsbereich zu melden.

CLI-Befehl:

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

Datei: `temperatureExample.json`

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

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

Antwort:

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

Example 3

Verwenden Sie den `BatchPutMessage` Vorgang, um eine `"seedTemperatureInput"` Nachricht zu senden, um den Wert der gewünschten Temperatur für einen bestimmten Bereich zu ändern.

CLI-Befehl:

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

Datei: `seedSetDesiredTemp.json`

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

Antwort:

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

MQTT-Nachrichten aufnehmen in AWS IoT Events

Wenn Ihre Sensordatenverarbeitungsressourcen die "BatchPutMessage" API nicht verwenden können, aber ihre Daten mithilfe eines einfachen MQTT-Clients an den AWS IoT Core Message Broker senden können, können Sie eine AWS IoT Core Themenregel erstellen, um Nachrichtendaten an eine AWS IoT Events Eingabe umzuleiten. Im Folgenden finden Sie eine Definition einer AWS IoT Events Themenregel, die die Eingabefelder "areaId" und die "sensorId" Eingabefelder aus dem MQTT-Thema und das "sensorData.temperature" Feld aus dem "temp" Nachrichten-Payload-Feld verwendet und diese Daten in unsere aufnimmt. AWS IoT Events "temperatureInput"

CLI-Befehl:

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

Datei: seedSetDesiredTemp.json

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

Antwort: [keine]

Wenn der Sensor eine Nachricht zum Thema "update/temperature/Area51/03" mit der folgenden Nutzlast sendet.

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

Dies führt dazu, dass Daten aufgenommen werden, AWS IoT Events als ob der folgende `"BatchPutMessage"` API-Aufruf getätigt worden wäre.

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

Datei: `spoofExample.json`

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

Generieren Sie Amazon SNS SNS-Nachrichten in AWS IoT Events

Im Folgenden finden Sie Beispiele für SNS-Nachrichten, die von der `"Area51"` Detector-Instance generiert wurden.

AWS IoT Events kann in Amazon SNS integriert werden, um Benachrichtigungen auf der Grundlage erkannter Ereignisse zu generieren und zu veröffentlichen. In diesem Abschnitt wird gezeigt, wie eine AWS IoT Events Detector-Instance, insbesondere der Detektor „Area51“, Amazon SNS SNS-Nachrichten generiert. Diese Beispiele veranschaulichen die Struktur und den Inhalt von Amazon SNS-Benachrichtigungen, die durch verschiedene Zustände und Ereignisse innerhalb des AWS IoT Events Melders ausgelöst werden, und veranschaulichen die Vorteile der Kombination AWS IoT Events mit Amazon SNS für Warnmeldungen und Kommunikation in Echtzeit.

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

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

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

Konfigurieren Sie die API in DescribeDetector AWS IoT Events

AWS IoT Events Mit dem `DescribeDetector` API-Eingang können Sie detaillierte Informationen zu einer bestimmten Detektorinstanz abrufen. Dieser Vorgang bietet Einblicke in den aktuellen Status, die Variablenwerte und die aktiven Timer eines Melders. Mithilfe dieser API können Sie den Status Ihrer AWS IoT Events Melder in Echtzeit überwachen und so das Debuggen, Analysieren und Verwalten Ihrer Workflows zur IoT-Ereignisverarbeitung erleichtern.

CLI-Befehl:

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

Antwort:

```
{
    "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"
    }
}
```

Verwenden Sie die AWS IoT Core Regel-Engine für AWS IoT Events

Die folgenden Regeln veröffentlichen AWS IoT Core MQTT-Nachrichten erneut als Shadow-Update-Anforderungsnachrichten. Wir gehen davon aus, dass für jeden Bereich, der durch das Detektormodell gesteuert wird, AWS IoT Core Dinge für eine Heiz- und eine Kühleinheit definiert sind. In diesem Beispiel haben wir Dinge mit dem Namen `"Area51HeatingUnit"` und definiert`"Area51CoolingUnit"`.

CLI-Befehl:

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

Datei: `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"
        }
      }
    ]
  }
}
```

Antwort: [leer]

CLI-Befehl:

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

Datei: `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"
        }
      }
    ]
```

```
  }
}
```

Antwort: [leer]

CLI-Befehl:

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

Datei: 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"
        }
      }
    ]
  }
}
```

Antwort: [leer]

CLI-Befehl:

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

Datei: 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"
        }
      }
    ]
  }
}
```

Antwort: [leer]

Unterstützte Aktionen zum Empfangen von Daten und Auslösen von Aktionen in AWS IoT Events

AWS IoT Events kann Aktionen auslösen, wenn ein bestimmtes Ereignis oder ein Übergangsereignis erkannt wird. Sie können integrierte Aktionen definieren, um einen Timer zu verwenden, eine Variable festzulegen oder Daten an andere AWS Ressourcen zu senden. Erfahren Sie, wie Sie diese Aktionen konfigurieren und anpassen können, um automatisierte Antworten auf Ihre verschiedenen IoT-Ereignisse zu erstellen.

> **Note**
>
> Wenn Sie eine Aktion in einem Detektormodell definieren, können Sie Ausdrücke für Parameter vom Datentyp Zeichenfolge verwenden. Weitere Informationen finden Sie unter Ausdrücke.

AWS IoT Events unterstützt die folgenden Aktionen, mit denen Sie einen Timer verwenden oder eine Variable setzen können:

- `setTimer`um einen Timer zu erstellen.
- `resetTimer`um den Timer zurückzusetzen.
- `clearTimer`um den Timer zu löschen.
- `setVariable`um eine Variable zu erstellen.

AWS IoT Events unterstützt die folgenden Aktionen, mit denen Sie mit AWS Diensten arbeiten können:

- `iotTopicPublish`um eine Nachricht zu einem MQTT-Thema zu veröffentlichen.
- `iotEvents`um Daten AWS IoT Events als Eingabewert an zu senden.
- `iotSiteWise` zum Senden von Daten an eine Komponenteneigenschaft in AWS IoT SiteWise
- `dynamoDB`um Daten an eine Amazon DynamoDB-Tabelle zu senden.
- `dynamoDBv2`um Daten an eine Amazon DynamoDB-Tabelle zu senden.
- `firehose`um Daten an einen Amazon Data Firehose-Stream zu senden.
- `lambda`um eine Funktion aufzurufen. AWS Lambda

- `sns`um Daten als Push-Benachrichtigung zu senden.
- `sqs`um Daten an eine Amazon SQS SQS-Warteschlange zu senden.

Verwenden Sie den AWS IoT Events integrierten Timer und die variablen Aktionen

AWS IoT Events unterstützt die folgenden Aktionen, mit denen Sie einen Timer verwenden oder eine Variable setzen können:

- `setTimer`um einen Timer zu erstellen.
- `resetTimer`um den Timer zurückzusetzen.
- `clearTimer`um den Timer zu löschen.
- `setVariable`um eine Variable zu erstellen.

Stellen Sie die Timer-Aktion ein

Set timer action

Mit `setTimer` dieser Aktion können Sie einen Timer mit einer Dauer in Sekunden erstellen.

More information (2)

Wenn Sie einen Timer erstellen, müssen Sie die folgenden Parameter angeben.

`timerName`

Der Name des Timers.

`durationExpression`

(Optional) Die Dauer des Timers in Sekunden.

Das ausgewertete Ergebnis eines Ausdrucks für die Dauer wird auf die nächste ganze Zahl abgerundet. Wenn Sie den Timer beispielsweise auf 60,99 Sekunden setzen, beträgt das ausgewertete Ergebnis des Ausdrucks für die Dauer 60 Sekunden.

Weitere Informationen finden Sie unter SetTimerAction in der AWS IoT Events -API-Referenz.

Timer-Aktion zurücksetzen

Reset timer action

Mit dieser `resetTimer` Aktion können Sie den Timer auf das zuvor ausgewertete Ergebnis des Dauerausdrucks setzen.

More information (1)

Wenn Sie einen Timer zurücksetzen, müssen Sie den folgenden Parameter angeben.

`timerName`

Der Name des Timers.

AWS IoT Events bewertet den Ausdruck für die Dauer nicht neu, wenn Sie den Timer zurücksetzen.

Weitere Informationen finden Sie unter ResetTimerAction in der AWS IoT Events -API-Referenz.

Löscht die Timer-Aktion

Clear timer action

Mit `clearTimer` dieser Aktion können Sie einen vorhandenen Timer löschen.

More information (1)

Wenn Sie einen Timer löschen, müssen Sie den folgenden Parameter angeben.

`timerName`

Der Name des Timers.

Weitere Informationen finden Sie unter ClearTimerAction in der AWS IoT Events -API-Referenz.

Legen Sie die variable Aktion fest

Set variable action

Mit der `setVariable` Aktion können Sie eine Variable mit einem bestimmten Wert erstellen.

More information (2)

Wenn Sie eine Variable erstellen, müssen Sie die folgenden Parameter angeben.

`variableName`

Der Name der Variable.

`value`

Der neue Wert der Variable.

Weitere Informationen finden Sie unter SetVariableAction in der AWS IoT Events -API-Referenz.

AWS IoT Events mit anderen AWS Diensten arbeiten

AWS IoT Events unterstützt die folgenden Aktionen, mit denen Sie mit AWS Diensten arbeiten können:

- `iotTopicPublish`um eine Nachricht zu einem MQTT-Thema zu veröffentlichen.
- `iotEvents`um Daten AWS IoT Events als Eingabewert an zu senden.
- `iotSiteWise` zum Senden von Daten an eine Komponenteneigenschaft in AWS IoT SiteWise
- `dynamoDB`um Daten an eine Amazon DynamoDB-Tabelle zu senden.
- `dynamoDBv2`um Daten an eine Amazon DynamoDB-Tabelle zu senden.
- `firehose`um Daten an einen Amazon Data Firehose-Stream zu senden.
- `lambda`um eine Funktion aufzurufen. AWS Lambda
- `sns`um Daten als Push-Benachrichtigung zu senden.
- `sqs`um Daten an eine Amazon SQS SQS-Warteschlange zu senden.

Important

- Sie müssen für beide AWS IoT Events und für die AWS Dienste, mit denen Sie arbeiten möchten, dieselbe AWS Region auswählen. Eine vollständige Liste der unterstützten Regionen finden Sie unter AWS IoT Events -Endpunkte und -Kontingente in Allgemeine Amazon Web Services-Referenz.

- Sie müssen dieselbe AWS Region verwenden, wenn Sie andere AWS Ressourcen für die AWS IoT Events Aktionen erstellen. Wenn Sie die AWS Region wechseln, haben Sie möglicherweise Probleme beim Zugriff auf die AWS Ressourcen.

AWS IoT Events Generiert standardmäßig eine Standardnutzlast in JSON für jede Aktion. Diese Aktionsnutzlast enthält alle Attribut-Wert-Paare, die Informationen über die Detektormodellinstanz und das Ereignis enthalten, das die Aktion ausgelöst hat. Um die Aktions-Payload zu konfigurieren, können Sie einen Inhaltsausdruck verwenden. Weitere Informationen finden Sie unter Ausdrücke zum Filtern, Transformieren und Verarbeiten von Ereignisdaten und zum Payload-Datentyp in der AWS IoT Events API-Referenz.

AWS IoT Core

IoT topic publish action

Mit dieser AWS IoT Core Aktion können Sie eine MQTT-Nachricht über den AWS IoT Message Broker veröffentlichen. Eine vollständige Liste der unterstützten Regionen finden Sie unter AWS IoT Core -Endpunkte und -Kontingente in Allgemeine Amazon Web Services-Referenz.

Der AWS IoT Message Broker verbindet AWS IoT Clients, indem er Nachrichten von Publishing-Clients an abonnierte Clients sendet. Weitere Informationen finden Sie unter Gerätekommunikationsprotokolle im AWS IoT Entwicklerhandbuch.

More information (2)

Wenn Sie eine MQTT-Nachricht veröffentlichen, müssen Sie die folgenden Parameter angeben.

`mqttTopic`

Das MQTT-Thema, das die Nachricht empfängt.

Sie können einen MQTT-Themennamen zur Laufzeit dynamisch definieren, indem Sie Variablen oder Eingabewerte verwenden, die im Detektormodell erstellt wurden.

`payload`

(Optional) Die Standard-Payload enthält alle Attribut-Wert-Paare, die Informationen über die Detektormodellinstanz und das Ereignis enthalten, das die Aktion ausgelöst hat. Sie können auch die Nutzlast anpassen. Weitere Informationen finden Sie unter Payload in der API-Referenz.AWS IoT Events

> **Note**
>
> Stellen Sie sicher, dass die mit Ihrer AWS IoT Events Servicerolle verknüpfte Richtlinie die `iot:Publish` Genehmigung erteilt. Weitere Informationen finden Sie unter Identitäts- und Zugriffsmanagement für AWS IoT Events.

Weitere Informationen finden Sie unter IotTopicPublishAction in der AWS IoT Events -API-Referenz.

AWS IoT Events

IoT Events action

Mit AWS IoT Events dieser Aktion können Sie Daten AWS IoT Events als Eingabe an senden. Eine vollständige Liste der unterstützten Regionen finden Sie unter AWS IoT Events -Endpunkte und -Kontingente in Allgemeine Amazon Web Services-Referenz.

AWS IoT Events ermöglicht es Ihnen, Ihre Geräte oder Geräteflotten auf Ausfälle oder Betriebsänderungen zu überwachen und bei Auftreten solcher Ereignisse Aktionen auszulösen. Weitere Informationen finden Sie unter Was ist AWS IoT Events? im AWS IoT Events Entwicklerhandbuch.

More information (2)

Wenn Sie Daten an senden AWS IoT Events, müssen Sie die folgenden Parameter angeben.

`inputName`

Der Name der AWS IoT Events Eingabe, die die Daten empfängt.

`payload`

(Optional) Die Standardnutzlast enthält alle Attribut-Wert-Paare, die Informationen über die Detektormodellinstanz und das Ereignis enthalten, das die Aktion ausgelöst hat. Sie können auch die Nutzlast anpassen. Weitere Informationen finden Sie unter Payload in der API-Referenz.AWS IoT Events

> **Note**
>
> Stellen Sie sicher, dass die mit Ihrer AWS IoT Events Servicerolle verknüpfte Richtlinie die `iotevents:BatchPutMessage` Genehmigung erteilt. Weitere Informationen finden Sie unter Identitäts- und Zugriffsmanagement für AWS IoT Events.

Weitere Informationen finden Sie unter IotEventsAction in der AWS IoT Events -API-Referenz.

AWS IoT SiteWise

IoT SiteWise action

Mit AWS IoT SiteWise dieser Aktion können Sie Daten an eine Anlageeigenschaft in senden AWS IoT SiteWise. Eine vollständige Liste der unterstützten Regionen finden Sie unter AWS IoT SiteWise -Endpunkte und -Kontingente in Allgemeine Amazon Web Services-Referenz.

AWS IoT SiteWise ist ein verwalteter Service, mit dem Sie Daten von Industrieanlagen in großem Umfang sammeln, organisieren und analysieren können. Weitere Informationen finden Sie unter Was ist AWS IoT SiteWise? im AWS IoT SiteWise -Benutzerhandbuch.

More information (11)

Wenn Sie Daten an ein Objekt in senden AWS IoT SiteWise, müssen Sie die folgenden Parameter angeben.

> **Important**
>
> Um die Daten zu empfangen, müssen Sie eine vorhandene Anlageneigenschaft in verwenden AWS IoT SiteWise.
>
> - Wenn Sie die AWS IoT Events Konsole verwenden, müssen Sie angeben`propertyAlias`, ob die Ziel-Asset-Eigenschaft identifiziert werden soll.
> - Wenn Sie die verwenden AWS CLI, müssen Sie eine `propertyAlias` oder beide angeben `assetId` und `propertyId` die Ziel-Asset-Eigenschaft identifizieren.
>
> Weitere Informationen finden Sie unter Zuordnung von industriellen Datenströmen zu Komponenten-Eigenschaften im Benutzerhandbuch für AWS IoT SiteWise .

propertyAlias

(Optional) Der Alias der Anlageneigenschaft. Sie können auch einen Ausdruck angeben.

assetId

(Optional) Die ID des Assets, das die angegebene Eigenschaft besitzt. Sie können auch einen Ausdruck angeben.

propertyId

(Optional) Die ID der Vermögenseigenschaft. Sie können auch einen Ausdruck angeben.

entryId

(Optional) Ein eindeutiger Bezeichner für diesen Eintrag. Sie können die Eintrags-ID verwenden, um zu verfolgen, welcher Dateneintrag im Fehlerfall einen Fehler verursacht. Der Standardwert ist ein neuer eindeutiger Bezeichner. Sie können auch einen Ausdruck angeben.

propertyValue

Eine Struktur, die Details zum Immobilienwert enthält.

quality

(Optional) Die Qualität des Vermögenswerts. Der Wert muss `GOOD`, `BAD` oder `UNCERTAIN` lauten. Sie können auch einen Ausdruck angeben.

timestamp

(Optional) Eine Struktur, die Zeitstempelinformationen enthält. Wenn Sie diesen Wert nicht angeben, ist die Standardeinstellung die Uhrzeit des Ereignisses.

timeInSeconds

Der Zeitstempel (in Sekunden) im Unix-Epoch-Format. Der gültige Bereich liegt zwischen 1-31556889864403199. Sie können auch einen Ausdruck angeben.

offsetInNanos

(Optional) Der Nanosekunden-Offset, konvertiert von`timeInSeconds`. Der gültige Bereich liegt zwischen 0-999999999. Sie können auch einen Ausdruck angeben.

value

Eine Struktur, die einen Asset-Eigenschaftswert enthält.

> **Important**
>
> Sie müssen je nach `dataType` der angegebenen Asset-Eigenschaft einen der folgenden Werttypen angeben. Weitere Informationen finden Sie unter AssetProperty in der AWS IoT SiteWise -API-Referenz.

`booleanValue`

(Optional) Der Eigenschaftswert der Anlage ist ein boolescher Wert, der oder sein muss. `TRUE` `FALSE` Sie können auch einen Ausdruck angeben. Wenn Sie einen Ausdruck verwenden, sollte das ausgewertete Ergebnis ein boolescher Wert sein.

`doubleValue`

(Optional) Der Wert der Anlageneigenschaft ist doppelt so hoch. Sie können auch einen Ausdruck angeben. Wenn Sie einen Ausdruck verwenden, sollte das ausgewertete Ergebnis ein Double sein.

`integerValue`

(Optional) Der Wert der Anlageneigenschaft ist eine Ganzzahl. Sie können auch einen Ausdruck angeben. Wenn Sie einen Ausdruck verwenden, sollte das ausgewertete Ergebnis eine Ganzzahl sein.

`stringValue`

(Optional) Der Wert der Anlageneigenschaft ist eine Zeichenfolge. Sie können auch einen Ausdruck angeben. Wenn Sie einen Ausdruck verwenden, sollte das ausgewertete Ergebnis eine Zeichenfolge sein.

> **Note**
>
> Vergewissern Sie sich, dass die mit Ihrer AWS IoT Events Servicerolle verknüpfte Richtlinie die `iotsitewise:BatchPutAssetPropertyValue` entsprechende Berechtigung erteilt. Weitere Informationen finden Sie unter Identitäts- und Zugriffsmanagement für AWS IoT Events.

Weitere Informationen finden Sie unter IotSiteWiseAction in der AWS IoT Events -API-Referenz.

Amazon DynamoDB

DynamoDB action

Mit der Amazon DynamoDB DynamoDB-Aktion können Sie Daten an eine DynamoDB-Tabelle senden. Eine Spalte der DynamoDB-Tabelle empfängt alle Attribut-Wert-Paare in der Aktions-Payload, die Sie angeben. Eine Liste der unterstützten Regionen finden Sie unter Amazon DynamoDB DynamoDB-Endpunkte und Kontingente in der. Allgemeine Amazon Web Services-Referenz

Amazon DynamoDB ist ein vollständig verwalteter NoSQL-Datenbank-Service, der schnelle und planbare Leistung mit nahtloser Skalierbarkeit bereitstellt. Weitere Informationen finden Sie unter Was ist DynamoDB? im Amazon DynamoDB DynamoDB-Entwicklerhandbuch.

More information (10)

Wenn Sie Daten an eine Spalte einer DynamoDB-Tabelle senden, müssen Sie die folgenden Parameter angeben.

`tableName`

Der Name der DynamoDB-Tabelle, die die Daten empfängt. Der `tableName` Wert muss mit dem Tabellennamen der DynamoDB-Tabelle übereinstimmen. Sie können auch einen Ausdruck angeben.

`hashKeyField`

Der Name des Hash-Schlüssels (auch Partitionsschlüssel genannt). Der `hashKeyField` Wert muss mit dem Partitionsschlüssel der DynamoDB-Tabelle übereinstimmen. Sie können auch einen Ausdruck angeben.

`hashKeyType`

(Optional) Der Datentyp des Hash-Schlüssels. Der Wert des Hash-Schlüsseltyps muss `STRING` oder sein`NUMBER`. Der Standardwert ist `STRING`. Sie können auch einen Ausdruck angeben.

`hashKeyValue`

Der Wert des Hash-Schlüssels Der `hashKeyValue` verwendet Ersatzvorlagen. Diese Vorlagen stellen Daten zur Laufzeit bereit. Sie können auch einen Ausdruck angeben.

`rangeKeyField`

(Optional) Der Name des Bereichsschlüssels (auch Sortierschlüssel genannt). Der `rangeKeyField` Wert muss mit dem Sortierschlüssel der DynamoDB-Tabelle übereinstimmen. Sie können auch einen Ausdruck angeben.

`rangeKeyType`

(Optional) Der Datentyp des Bereichsschlüssels. Der Wert des Hash-Schlüsseltyps muss `STRING` oder sein`NUMBER`. Der Standardwert ist `STRING`. Sie können auch einen Ausdruck angeben.

`rangeKeyValue`

(Optional) Der Wert des Bereichsschlüssels. Der `rangeKeyValue` verwendet Ersatzvorlagen. Diese Vorlagen stellen Daten zur Laufzeit bereit. Sie können auch einen Ausdruck angeben.

operation

(Optional) Die Art des auszuführenden Vorgangs. Sie können auch einen Ausdruck angeben. Der Operationswert muss einer der folgenden Werte sein:

- `INSERT` – Fügt Daten als neues Element in die DynamoDB-Tabelle ein. Dies ist der Standardwert.
- `UPDATE` – Aktualisiert ein vorhandenes Element der DynamoDB-Tabelle mit neuen Daten.
- `DELETE`- Löscht ein vorhandenes Element aus der DynamoDB-Tabelle.

`payloadField`

(Optional) Der Name der DynamoDB-Spalte, die die Aktions-Payload empfängt. Der Standardname lautet `payload`. Sie können auch einen Ausdruck angeben.

`payload`

(Optional) Die Standardnutzlast enthält alle Attribut-Wert-Paare, die Informationen über die Detektormodellinstanz und das Ereignis enthalten, das die Aktion ausgelöst hat. Sie können auch die Nutzlast anpassen. Weitere Informationen finden Sie unter Payload in der API-Referenz.AWS IoT Events

Wenn der angegebene Payload-Typ eine Zeichenfolge ist, werden Nicht-JSON-Daten als Binärdaten an die DynamoDB-Tabelle `DynamoDBAction` gesendet. Die DynamoDB-Konsole zeigt die Daten als Base64-encoded Text an. Der Wert von `payloadField` ist *`payload-field`*`_raw`. Sie können auch einen Ausdruck angeben.

> **Note**
>
> Stellen Sie sicher, dass die mit Ihrer AWS IoT Events Servicerolle verknüpfte Richtlinie die `dynamodb:PutItem` Berechtigung erteilt. Weitere Informationen finden Sie unter Identitäts- und Zugriffsmanagement für AWS IoT Events.

Weitere Informationen finden Sie unter DynamoDBAction in der AWS IoT Events API-Referenz.

Amazon DynamoDB (v2)

DynamoDBv2 action

Mit der Amazon DynamoDB (v2) -Aktion können Sie Daten in eine DynamoDB-Tabelle schreiben. Eine separate Spalte der DynamoDB-Tabelle erhält ein Attribut-Wert-Paar in der Aktions-Payload, die Sie angeben. Eine Liste der unterstützten Regionen finden Sie unter Amazon DynamoDB DynamoDB-Endpunkte und Kontingente in der. Allgemeine Amazon Web Services-Referenz

Amazon DynamoDB ist ein vollständig verwalteter NoSQL-Datenbank-Service, der schnelle und planbare Leistung mit nahtloser Skalierbarkeit bereitstellt. Weitere Informationen finden Sie unter Was ist DynamoDB? im Amazon DynamoDB DynamoDB-Entwicklerhandbuch.

More information (2)

Wenn Sie Daten an mehrere Spalten einer DynamoDB-Tabelle senden, müssen Sie die folgenden Parameter angeben.

`tableName`

Der Name der DynamoDB-Tabelle, die die Daten empfängt. Sie können auch einen Ausdruck angeben.

`payload`

(Optional) Die Standardnutzlast enthält alle Attribut-Wert-Paare, die Informationen über die Detektormodellinstanz und das Ereignis enthalten, das die Aktion ausgelöst hat. Sie können auch die Nutzlast anpassen. Weitere Informationen finden Sie unter Payload in der API-Referenz.AWS IoT Events

> **Important**
>
> Der Payload-Typ muss JSON sein. Sie können auch einen Ausdruck angeben.

> **Note**
>
> Stellen Sie sicher, dass die mit Ihrer AWS IoT Events Servicerolle verknüpfte Richtlinie die `dynamodb:PutItem` Genehmigung erteilt. Weitere Informationen finden Sie unter Identitäts- und Zugriffsmanagement für AWS IoT Events.

Weitere Informationen finden Sie unter DynamoDBv2Action in der AWS IoT Events -API-Referenz.

Amazon Data Firehose

Firehose action

Mit der Amazon Data Firehose-Aktion können Sie Daten an einen Firehose-Lieferstream senden. Eine Liste der unterstützten Regionen finden Sie unter Amazon Data Firehose Endpoints and Quotas in der. Allgemeine Amazon Web Services-Referenz

Amazon Data Firehose ist ein vollständig verwalteter Service für die Bereitstellung von Echtzeit-Streaming-Daten an Ziele wie Amazon Simple Storage Service (Amazon Simple Storage Service), Amazon Redshift, Amazon OpenSearch Service (OpenSearch Service) und Splunk. Weitere Informationen finden Sie unter Was ist Amazon Data Firehose? im Amazon Data Firehose Developer Guide.

More information (3)

Wenn Sie Daten an einen Firehose-Lieferstream senden, müssen Sie die folgenden Parameter angeben.

`deliveryStreamName`

Der Name des Firehose-Lieferstreams, der die Daten empfängt.

`separator`

(Optional) Sie können ein Zeichentrennzeichen verwenden, um fortlaufende Daten zu trennen, die an den Firehose-Lieferstream gesendet werden. Der Wert des Trennzeichens muss `'\n'` (neue Zeile), `'\t'` (Tab), `'\r\n'` (Windows-neue Zeile) oder `','` (Komma) lauten.

`payload`

(Optional) Die Standardnutzlast enthält alle Attribut-Wert-Paare, die Informationen über die Detektormodellinstanz und das Ereignis enthalten, das die Aktion ausgelöst hat. Sie können auch die Nutzlast anpassen. Weitere Informationen finden Sie unter Payload in der API-Referenz.AWS IoT Events

> **Note**
>
> Stellen Sie sicher, dass die mit Ihrer AWS IoT Events Servicerolle verknüpfte Richtlinie die `firehose:PutRecord` Genehmigung erteilt. Weitere Informationen finden Sie unter Identitäts- und Zugriffsmanagement für AWS IoT Events.

Weitere Informationen finden Sie unter FirehoseAction in der AWS IoT Events -API-Referenz.

AWS Lambda

Lambda action

Mit der AWS Lambda Aktion können Sie eine Lambda-Funktion aufrufen. Eine vollständige Liste der unterstützten Regionen finden Sie unter AWS Lambda -Endpunkte und -Kontingente in Allgemeine Amazon Web Services-Referenz.

AWS Lambda ist ein Rechendienst, mit dem Sie Code ausführen können, ohne Server bereitzustellen oder zu verwalten. Weitere Informationen finden Sie unter Was ist AWS Lambda? im AWS Lambda Entwicklerhandbuch.

More information (2)

Wenn Sie eine Lambda-Funktion aufrufen, müssen Sie die folgenden Parameter angeben.

`functionArn`

Der ARN der aufzurufenden Lambda-Funktion.

`payload`

(Optional) Die Standardnutzlast enthält alle Attribut-Wert-Paare, die Informationen über die Detektormodellinstanz und das Ereignis enthalten, das die Aktion ausgelöst hat. Sie können auch die Nutzlast anpassen. Weitere Informationen finden Sie unter Payload in der API-Referenz.AWS IoT Events

> **Note**
>
> Stellen Sie sicher, dass die mit Ihrer AWS IoT Events Servicerolle verknüpfte Richtlinie die `lambda:InvokeFunction` Genehmigung erteilt. Weitere Informationen finden Sie unter Identitäts- und Zugriffsmanagement für AWS IoT Events.

Weitere Informationen finden Sie unter LambdaAction in der AWS IoT Events -API-Referenz.

Amazon Simple Notification Service

SNS action

Mit der Aktion zum Veröffentlichen von Amazon SNS SNS-Themen können Sie eine Amazon SNS SNS-Nachricht veröffentlichen. Eine Liste der unterstützten Regionen finden Sie unter Amazon Simple Notification Service-Endpunkte und Kontingente in der Allgemeine Amazon Web Services-Referenz.

Amazon Simple Notification Service (Amazon Simple Notification Service) ist ein Webservice, der die Zustellung oder den Versand von Nachrichten an abonnierte Endpunkte oder Kunden koordiniert und verwaltet. Weitere Informationen finden Sie unter Was ist Amazon SNS? im Amazon Simple Notification Service Developer Guide.

> **Note**
>
> Die Aktion zur Veröffentlichung von Amazon SNS SNS-Themen unterstützt keine Amazon SNS SNS-FIFO-Themen (first in, first out). Da es sich bei der Regel-Engine um einen vollständig verteilten Service handelt, werden die Nachrichten möglicherweise nicht in der angegebenen Reihenfolge angezeigt, wenn die Amazon SNS SNS-Aktion initiiert wird.

More information (2)

Wenn Sie eine Amazon SNS SNS-Nachricht veröffentlichen, müssen Sie die folgenden Parameter angeben.

`targetArn`

Der ARN des Amazon SNS-Ziels, das die Nachricht empfängt.

`payload`

(Optional) Die Standard-Payload enthält alle Attribut-Wert-Paare, die Informationen über die Detector-Modell-Instance und das Ereignis enthalten, das die Aktion ausgelöst hat. Sie können auch die Nutzlast anpassen. Weitere Informationen finden Sie unter Payload in der API-Referenz.AWS IoT Events

> **Note**
>
> Stellen Sie sicher, dass die mit Ihrer AWS IoT Events Servicerolle verknüpfte Richtlinie die `sns:Publish` Genehmigung erteilt. Weitere Informationen finden Sie unter Identitäts- und Zugriffsmanagement für AWS IoT Events.

Weitere Informationen finden Sie unter SNSTopicPublishAction in der AWS IoT Events -API-Referenz.

Amazon Simple Queue Service

SQS action

Mit der Amazon SQS SQS-Aktion können Sie Daten an eine Amazon SQS SQS-Warteschlange senden. Eine Liste der unterstützten Regionen finden Sie unter Amazon Simple Queue Service-Endpunkte und Kontingente in der Allgemeine Amazon Web Services-Referenz.

Amazon Simple Queue Service (Amazon SQS) bietet eine sichere, dauerhafte und verfügbare gehostete Warteschlange, die es Ihnen ermöglicht, verteilte Softwaresysteme und -komponenten zu integrieren und zu entkoppeln. Weitere Informationen finden Sie unter Was ist Amazon Simple Queue Service> im Amazon Simple Queue Service Developer Guide.

> **Note**
>
> Die Amazon SQS-Aktion unterstützt keine >Amazon SQS FIFO (first in, first out) - Themen. Da es sich bei der Regel-Engine um einen vollständig verteilten Service handelt, werden die Nachrichten möglicherweise nicht in der angegebenen Reihenfolge angezeigt, wenn die Amazon SQS SQS-Aktion initiiert wird.

More information (3)

Wenn Sie Daten an eine Amazon SQS SQS-Warteschlange senden, müssen Sie die folgenden Parameter angeben.

`queueUrl`

Die URL der Amazon SQS SQS-Warteschlange, die die Daten empfängt.

`useBase64`

(Optional) AWS IoT Events Kodiert die Daten in Base64-Text, sofern Sie dies angeben. `TRUE` Der Standardwert ist `FALSE`.

`payload`

(Optional) Die Standardnutzlast enthält alle Attribut-Wert-Paare, die Informationen über die Detektormodellinstanz und das Ereignis enthalten, das die Aktion ausgelöst hat. Sie können auch die Nutzlast anpassen. Weitere Informationen finden Sie unter Payload in der API-Referenz.AWS IoT Events

> **Note**
>
> Stellen Sie sicher, dass die mit Ihrer AWS IoT Events Servicerolle verknüpfte Richtlinie die `sqs:SendMessage` Genehmigung erteilt. Weitere Informationen finden Sie unter Identitäts- und Zugriffsmanagement für AWS IoT Events.

Weitere Informationen finden Sie unter SNSTopicPublishAction in der AWS IoT Events -API-Referenz.

Sie können auch Amazon SNS und die AWS IoT Core Rules Engine verwenden, um eine AWS Lambda Funktion auszulösen. Dies ermöglicht es, Maßnahmen mithilfe anderer Dienste wie Connect Customer oder sogar einer ERP-Anwendung (Enterprise Resource Planning) für Unternehmen zu ergreifen.

> **Note**
>
> Um große Datenströme in Echtzeit zu sammeln und zu verarbeiten, können Sie andere AWS Dienste wie Amazon Kinesis verwenden. Von dort aus können Sie eine erste Analyse durchführen und die Ergebnisse dann AWS IoT Events als Eingabe an einen Detektor senden.

Ausdrücke zum Filtern, Transformieren und Verarbeiten von Ereignisdaten

Ausdrücke werden verwendet, um eingehende Daten auszuwerten, Berechnungen durchzuführen und die Bedingungen zu bestimmen, unter denen bestimmte Aktionen oder Zustandsübergänge stattfinden sollten. AWS IoT Events bietet mehrere Möglichkeiten, Werte anzugeben, wenn Sie Detektormodelle erstellen und aktualisieren. Sie können Ausdrücke verwenden, um Literalwerte anzugeben, oder Sie AWS IoT Events können die Ausdrücke auswerten, bevor Sie bestimmte Werte angeben.

Themen

Syntax zum Filtern von Gerätedaten und zum Definieren von Aktionen in AWS IoT Events

Ausdrücke bieten Syntax zum Filtern von Gerätedaten und zum Definieren von Aktionen. Sie können Literale, Operatoren, Funktionen, Verweise und Ersatzvorlagen in den AWS IoT Events Ausdrücken verwenden. Durch die Kombination dieser Komponenten können Sie leistungsstarke und flexible Ausdrücke erstellen, um IoT-Daten zu verarbeiten, Berechnungen durchzuführen, Zeichenketten zu manipulieren und logische Entscheidungen innerhalb Ihrer Detektormodelle zu treffen.

Literale

- Ganzzahl
- Dezimal
- Zeichenfolge
- Boolesch

Betreiber

Unär

- Nicht (Boolean): `!`

- Nicht (bitweise): ~
- Minus (Arithmetik): -

Zeichenfolge

- Verkettung: **+**

  Beide Operanden müssen Zeichenketten sein. Zeichenkettenliterale müssen in einfache Anführungszeichen (') eingeschlossen werden.

  Zum Beispiel: -> `'my' + 'string' 'mystring'`

Arithmetisch

- Zusatz: **+**

  Beide Operanden müssen numerisch sein.
- Subtraktion: **-**
- Einteilung: **/**

  Das Ergebnis der Division ist ein gerundeter Ganzzahlwert, sofern nicht mindestens einer der Operanden (Divisor oder Dividend) ein Dezimalwert ist.
- Multiplikation: *****

Bitweise (Ganzzahl)

- ODER: **|**

  Zum Beispiel: `13 | 5` -> `13`
- UND: **&**

  Zum Beispiel: `13 & 5` -> `5`
- XOR: **^**

  Zum Beispiel: -> `13 ^ 5 8`
- NICHT: **~**

  Zum Beispiel: `~13` -> `-14`

Boolesch

- Weniger als: **<**
- Weniger als oder gleich: **<=**
- Gleich: **==**

- Nicht gleich: **!=**
- Größer als oder gleich: **>=**
- Größer als: **>**
- UND: **&&**
- ODER: **||**

> **Note**
>
> Wenn ein Unterausdruck von undefinierte Daten `||` enthält, wird dieser Unterausdruck als behandelt. `false`

Klammern

Sie können Klammern verwenden, um Begriffe innerhalb eines Ausdrucks zu gruppieren.

Funktionen zur Verwendung in Ausdrücken AWS IoT Events

AWS IoT Events bietet eine Reihe integrierter Funktionen, um die Möglichkeiten Ihrer Detektormodellausdrücke zu erweitern. Diese Funktionen ermöglichen Timerverwaltung, Typkonvertierung, Nullprüfung, Triggertyp-Identifizierung, Eingabeüberprüfung, Zeichenkettenmanipulation und bitweise Operationen. Durch die Nutzung dieser Funktionen können Sie eine reaktionsschnelle AWS IoT Events Verarbeitungslogik erstellen und so die Gesamteffektivität Ihrer IoT-Anwendungen verbessern.

Integrierte Funktionen

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

Prüft, `true` ob der angegebene Timer abgelaufen ist. Ersetzen Sie "*`timer-name`*" durch den Namen eines von Ihnen definierten Timers in Anführungszeichen. In einer Ereignisaktion können Sie einen Timer definieren und dann den Timer starten, zurücksetzen oder einen zuvor definierten Timer löschen. Sehen Sie sich das Feld an`detectorModelDefinition.states.onInput|onEnter|onExit.events.actions.setTimer.timerName`.

Auf einen Timer, der in einem Status festgelegt ist, kann in einem anderen Status verwiesen werden. Sie müssen den Status besuchen, in dem Sie den Timer erstellt haben, bevor Sie den Status aufrufen, in dem auf den Timer verwiesen wird.

Ein Detektormodell hat beispielsweise zwei Zustände, `TemperatureChecked` und`RecordUpdated`. Sie haben im `TemperatureChecked` Bundesstaat einen Timer erstellt. Sie müssen den `TemperatureChecked` Bundesstaat zuerst besuchen, bevor Sie den Timer im `RecordUpdated` Bundesstaat verwenden können.

Um die Genauigkeit zu gewährleisten, sollte ein Timer mindestens 60 Sekunden eingestellt werden.

> **Note**
>
> `timeout()`gibt `true` nur das erste Mal zurück, wenn der Timer nach Ablauf des Timers zum ersten Mal überprüft wird, und kehrt `false` danach zurück.

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

Ergibt den Wert des Ausdrucks, der in den angegebenen Typ konvertiert wurde. Der *`type`* Wert muss `String`, `Boolean`, oder `Decimal` sein. Verwenden Sie eines dieser Schlüsselwörter oder einen Ausdruck, der eine Zeichenfolge ergibt, die das Schlüsselwort enthält. Nur die folgenden Konvertierungen sind erfolgreich und geben einen gültigen Wert zurück:

- Boolean -> Zeichenfolge

  Gibt die Zeichenfolge `"true"` oder zurück. `"false"`
- Dezimal -> Zeichenfolge
- Zeichenfolge -> Boolean
- Zeichenfolge -> Dezimalzahl

  Die angegebene Zeichenfolge muss eine gültige Darstellung einer Dezimalzahl sein, andernfalls schlägt sie `convert()` fehl.

Wenn `convert()` kein gültiger Wert zurückgegeben wird, ist der Ausdruck, zu dem er gehört, ebenfalls ungültig. Dieses Ergebnis entspricht dem Ereignis, in dem der `actions` Ausdruck auftritt, `false` und löst den Übergang zum `nextState` angegebenen Wert nicht aus.

`isNull`(*`expression`*)

Wird ausgewertet, `true` ob der Ausdruck Null zurückgibt. Wenn die Eingabe beispielsweise die Nachricht `MyInput` empfängt`{ "a": null }`, wird Folgendes als ausgewertet`true`, aber als `isUndefined($input.MyInput.a)` ausgewertet. `false`

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

`isUndefined`(*`expression`*)

Wird als undefiniert ausgewertet, `true` wenn der Ausdruck undefiniert ist. Wenn die Eingabe beispielsweise die Nachricht `MyInput` empfängt`{ "a": null }`, wird Folgendes als ausgewertet`false`, aber `isNull($input.MyInput.a)` als ausgewertet. `true`

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

`triggerType`("*`type`*")

Der *`type`* Wert kann oder sein. `"Message"` `"Timer"` Prüft, `true` ob die Ereignisbedingung, in der sie auftritt, ausgewertet wird, weil ein Timer abgelaufen ist, wie im folgenden Beispiel.

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

Oder es wurde eine Eingabenachricht empfangen.

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

`currentInput`("*`input`*")

Prüft, `true` ob die Ereignisbedingung, in der sie auftritt, ausgewertet wird, weil die angegebene Eingabemeldung empfangen wurde. Wenn die Eingabe beispielsweise die Nachricht `Command` empfängt`{ "value": "Abort" }`, wird Folgendes als ausgewertet. `true`

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

Verwenden Sie diese Funktion, um zu überprüfen, ob die Bedingung ausgewertet wird, weil eine bestimmte Eingabe empfangen wurde und ein Timer nicht abgelaufen ist, wie im folgenden Ausdruck.

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

Funktionen zum Abgleich von Zeichenketten

`startsWith`(*expression1*, *expression2*)

Prüft, `true` ob der erste Zeichenkettenausdruck mit dem zweiten Zeichenkettenausdruck beginnt. Wenn die Eingabe beispielsweise die Nachricht `MyInput` empfängt`{ "status": "offline"}`, wird Folgendes als ausgewertet. `true`

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

Beide Ausdrücke müssen einen Zeichenkettenwert ergeben. Wenn einer der Ausdrücke keinen Zeichenkettenwert ergibt, ist das Ergebnis der Funktion undefiniert. Es werden keine Konvertierungen durchgeführt.

`endsWith`(*expression1*, *expression2*)

Prüft, `true` ob der erste Zeichenkettenausdruck mit dem zweiten Zeichenkettenausdruck endet. Wenn die Eingabe beispielsweise die Nachricht `MyInput` empfängt`{ "status": "offline" }`, wird Folgendes als ausgewertet. `true`

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

Beide Ausdrücke müssen einen Zeichenkettenwert ergeben. Wenn einer der Ausdrücke keinen Zeichenkettenwert ergibt, ist das Ergebnis der Funktion undefiniert. Es werden keine Konvertierungen durchgeführt.

`contains`(*expression1*, *expression2*)

Prüft, `true` ob der erste Zeichenkettenausdruck den zweiten Zeichenkettenausdruck enthält. Wenn die Eingabe beispielsweise die Nachricht `MyInput` empfängt`{ "status": "offline" }`, wird Folgendes als ausgewertet. `true`

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

Beide Ausdrücke müssen einen Zeichenkettenwert ergeben. Wenn einer der Ausdrücke keinen Zeichenkettenwert ergibt, ist das Ergebnis der Funktion undefiniert. Es werden keine Konvertierungen durchgeführt.

Funktionen zur bitweisen Ganzzahlmanipulation

bitor(*expression1*, *expression2*)

Wertet das bitweise ODER der Integer-Ausdrücke aus (die binäre OR-Operation wird für die entsprechenden Bits der Ganzzahlen ausgeführt). Wenn die Eingabe beispielsweise die Nachricht `MyInput` empfängt`{ "value1": 13, "value2": 5 }`, wird Folgendes als ausgewertet. `13`

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

Beide Ausdrücke müssen einen ganzzahligen Wert ergeben. Wenn einer der Ausdrücke keinen ganzzahligen Wert ergibt, ist das Ergebnis der Funktion undefiniert. Es werden keine Konvertierungen durchgeführt.

bitand(*expression1*, *expression2*)

Wertet das bitweise UND der Integer-Ausdrücke aus (die binäre AND-Operation wird für die entsprechenden Bits der Ganzzahlen ausgeführt). Wenn die Eingabe beispielsweise die Nachricht `MyInput` empfängt`{ "value1": 13, "value2": 5 }`, wird Folgendes als ausgewertet. `5`

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

Beide Ausdrücke müssen einen ganzzahligen Wert ergeben. Wenn einer der Ausdrücke keinen ganzzahligen Wert ergibt, ist das Ergebnis der Funktion undefiniert. Es werden keine Konvertierungen durchgeführt.

bitxor(*expression1*, *expression2*)

Wertet das bitweise XOR der Integer-Ausdrücke aus (die binäre XOR-Operation wird für die entsprechenden Bits der Ganzzahlen ausgeführt). Wenn die Eingabe beispielsweise die Nachricht `MyInput` empfängt`{ "value1": 13, "value2": 5 }`, wird Folgendes als ausgewertet. `8`

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

Beide Ausdrücke müssen einen ganzzahligen Wert ergeben. Wenn einer der Ausdrücke keinen ganzzahligen Wert ergibt, ist das Ergebnis der Funktion undefiniert. Es werden keine Konvertierungen durchgeführt.

bitnot(*expression*)

Wertet das bitweise NOT des Integer-Ausdrucks aus (die binäre NOT-Operation wird für die Bits der Ganzzahl ausgeführt). Wenn die Eingabe beispielsweise die Nachricht `MyInput` empfängt`{ "value": 13 }`, wird Folgendes als ausgewertet. `-14`

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

Beide Ausdrücke müssen einen ganzzahligen Wert ergeben. Wenn einer der Ausdrücke keinen ganzzahligen Wert ergibt, ist das Ergebnis der Funktion undefiniert. Es werden keine Konvertierungen durchgeführt.

AWS IoT Events Referenz für Eingaben und Variablen in Ausdrücken

Eingaben

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

`input-name`ist eine Eingabe, die Sie mithilfe der CreateInputAktion erstellen.

Wenn Sie beispielsweise eine Eingabe benannt haben, `TemperatureInput` für die Sie `inputDefinition.attributes.jsonPath` Einträge definiert haben, werden die Werte möglicherweise in den folgenden verfügbaren Feldern angezeigt.

```
{
    "temperature": 78.5,
    "date": "2018-10-03T16:09:09Z"
  }
```

Verwenden Sie den folgenden Befehl, um `temperature` auf den Wert des Felds zu verweisen.

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

Bei Feldern, deren Werte Arrays sind, können Sie mit Hilfe `[`*`n`*`]` von auf Elemente des Arrays verweisen. Zum Beispiel bei den folgenden Werten:

```
{
    "temperatures": [
      78.4,
      77.9,
```

```
      78.8
    ],
    "date": "2018-10-03T16:09:09Z"
  }
```

Auf den Wert `78.8` kann mit dem folgenden Befehl verwiesen werden.

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

Variablen

`$variable.`*`variable-name`*

Das *`variable-name`* ist eine Variable, die Sie mithilfe der CreateDetectorModelAktion definiert haben.

Wenn Sie beispielsweise eine Variable benannt haben, `TechnicianID` die Sie mithilfe definiert habendetectorDefinition.states.onInputEvents.actions.setVariable.variableName, können Sie mit dem folgenden Befehl auf den Wert (Zeichenfolge) verweisen, der der Variablen zuletzt zugewiesen wurde.

```
$variable.TechnicianID
```

Sie können die Werte von Variablen nur mithilfe der `setVariable` Aktion festlegen. Sie können Variablen in einem Ausdruck keine Werte zuweisen. Eine Variable kann nicht rückgängig gemacht werden. Sie können ihr beispielsweise den Wert `null` nicht zuweisen.

> **Note**
>
> In Verweisen, die Bezeichner verwenden, die nicht dem Muster (regulärer Ausdruck) folgen`[a-zA-Z][a-zA-Z0-9_]*`, müssen Sie diese Bezeichner in Backticks () einschließen. ` Beispielsweise `_value` muss ein Verweis auf eine Eingabe, die `MyInput` mit einem Feld benannt ist, dieses Feld als angeben. `$input.MyInput.`_value``

Wenn Sie Verweise in Ausdrücken verwenden, überprüfen Sie Folgendes:

- Wenn Sie eine Referenz als Operanden mit einem oder mehreren Operatoren verwenden, stellen Sie sicher, dass alle Datentypen, auf die Sie verweisen, kompatibel sind.

Im folgenden Ausdruck 2 ist Integer beispielsweise ein Operand sowohl der == Operatoren als auch. && Um sicherzustellen, dass die Operanden kompatibel sind `$variable.testVariable + 1` und auf eine Ganzzahl oder Dezimalzahl verweisen `$variable.testVariable` müssen.

Außerdem 1 ist Integer ein Operand des Operators+. `$variable.testVariable`Muss daher auf eine Ganzzahl oder Dezimalzahl verweisen.

```
‘$variable.testVariable + 1 == 2 && $variable.testVariable’
```

- Wenn Sie eine Referenz als Argument verwenden, das an eine Funktion übergeben wird, stellen Sie sicher, dass die Funktion die Datentypen unterstützt, auf die Sie verweisen.

  Für die folgende `timeout("`*`time-name`*`")` Funktion ist beispielsweise eine Zeichenfolge mit doppelten Anführungszeichen als Argument erforderlich. Wenn Sie einen Verweis für den *`timer-name`* Wert verwenden, müssen Sie auf eine Zeichenfolge mit doppelten Anführungszeichen verweisen.

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

  > **Note**
  >
  > Wenn Sie für die `convert(`*`type, expression`*`)` Funktion eine Referenz für den *`type`* Wert verwenden, muss das ausgewertete Ergebnis Ihrer Referenz `String`Decimal, oder sein`Boolean`.

AWS IoT Events Ausdrücke unterstützen die Datentypen Integer, Decimal, String und Boolean. Die folgende Tabelle enthält eine Liste inkompatibler Typpaare.

Inkompatible Typenpaare
Ganzzahl, Zeichenfolge
Ganzzahl, Boolean
Dezimal, Zeichenfolge
Dezimal, Boolesch

Inkompatible Typenpaare
Zeichenfolge, Boolean

Substitutionsvorlagen für Ausdrücke AWS IoT Events

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

Der `${}` identifiziert die Zeichenfolge als interpolierte Zeichenfolge. Das `expression` kann ein beliebiger Ausdruck sein. AWS IoT Events Dazu gehören Operatoren, Funktionen und Verweise.

Sie haben die SetVariableActionAktion beispielsweise verwendet, um eine Variable zu definieren. `variableName` ist `SensorID` und `value` ist `10`. Sie können die folgenden Substitutionsvorlagen erstellen.

Substitutionsvorlage	Ergebniszeichenfolge
`'${'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}"`

Ausdrucksbeispiele und Verwendung für AWS IoT Events

Sie können Werte in einem Detektormodell auf folgende Weise angeben:

- Geben Sie unterstützte Ausdrücke in der AWS IoT Events Konsole ein.
- Übergeben Sie die Ausdrücke an die AWS IoT Events APIs AS-Parameter.

Ausdrücke unterstützen Literale, Operatoren, Funktionen, Verweise und Ersatzvorlagen.

> **Important**
>
> Ihre Ausdrücke müssen auf eine Ganzzahl, eine Dezimalzahl, eine Zeichenfolge oder einen booleschen Wert verweisen.

Ausdrücke schreiben AWS IoT Events

Sehen Sie sich die folgenden Beispiele an, die Ihnen beim Schreiben Ihrer AWS IoT Events Ausdrücke helfen sollen:

Literal

Bei Literalwerten müssen die Ausdrücke einfache Anführungszeichen enthalten. Ein boolescher Wert muss entweder oder `true` sein. `false`

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

Referenz

Bei Referenzen müssen Sie entweder Variablen oder Eingabewerte angeben.

- Die folgende Eingabe bezieht sich auf eine Dezimalzahl,. `10.01`

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

- Die folgende Variable verweist auf eine Zeichenfolge,`Greenhouse Temperature Table`.

  ```
  $variable.TableName
  ```

Vorlage für die Substitution

Für eine Substitutionsvorlage müssen Sie `${}` verwenden und die Vorlage muss von einfachen Anführungszeichen umschlossen sein. Eine Substitutionsvorlage kann auch eine Kombination aus Literalen, Operatoren, Funktionen, Referenzen und Substitutionsvorlagen enthalten.

- Das ausgewertete Ergebnis des folgenden Ausdrucks ist eine Zeichenfolge,`50.018 in Fahrenheit`.

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

- Das ausgewertete Ergebnis des folgenden Ausdrucks ist eine Zeichenfolge,`{\"sensor_id\":\"Sensor_1\",\"temperature\":\"50.018\"}`.

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

Zeichenfolgenverkettung

Für eine Zeichenfolgeverkettung müssen Sie + verwenden. Eine Zeichenfolgeverkettung kann auch eine Kombination aus Literalen, Operatoren, Funktionen, Referenzen und Substitutionsvorlagen enthalten.

- Das ausgewertete Ergebnis des folgenden Ausdrucks ist eine Zeichenfolge,`Greenhouse Temperature Table 2000-01-01`.

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

AWS IoT Events Beispiele für Detektormodelle

Diese Seite enthält eine Liste von Anwendungsbeispielen, die zeigen, wie verschiedene AWS IoT Events Funktionen konfiguriert werden. Die Beispiele reichen von grundlegenden Erkennungen wie Temperaturgrenzwerten bis hin zu komplexeren Szenarien zur Erkennung von Anomalien und maschinellem Lernen. Jedes Beispiel enthält Verfahren und Codefragmente, die Ihnen bei der Einrichtung von AWS IoT Events Erkennungen, Aktionen und Integrationen helfen. Diese Beispiele zeigen die Flexibilität des AWS IoT Events Dienstes und wie er für verschiedene IoT-Anwendungen und Anwendungsfälle angepasst werden kann. Auf dieser Seite finden Sie Informationen zu den AWS IoT Events Funktionen oder wenn Sie Hilfe bei der Implementierung eines bestimmten Erkennungs- oder Automatisierungsworkflows benötigen.

Topics

Beispiel: Verwendung der HVAC-Temperatursteuerung mit AWS IoT Events

Hintergrundgeschichte

In diesem Beispiel wird ein Temperaturregelungsmodell (ein Thermostat) mit folgenden Funktionen implementiert:

- Ein von Ihnen definiertes Meldermodell, das mehrere Bereiche überwachen und steuern kann. (Für jeden Bereich wird eine Melderinstanz erstellt.)

- Jede Melderinstanz empfängt Temperaturdaten von mehreren Sensoren, die sich in jedem Kontrollbereich befinden.
- Sie können die gewünschte Temperatur (den Sollwert) für jeden Bereich jederzeit ändern.
- Sie können die Betriebsparameter für jeden Bereich definieren und diese Parameter jederzeit ändern.
- Sie können jederzeit Sensoren zu einem Bereich hinzufügen oder Sensoren aus einem Bereich löschen.
- Sie können Heiz- und Kühlgeräte mit einer Mindestlaufzeit aktivieren, um sie vor Beschädigungen zu schützen.
- Die Melder weisen anomale Sensorwerte zurück und melden sie.
- Sie können Sollwerte für die Notfalltemperatur definieren. Wenn ein Sensor eine Temperatur über oder unter den von Ihnen definierten Sollwerten meldet, werden die Heiz- oder Kühlgeräte sofort eingeschaltet und der Melder meldet diese Temperaturspitze.

In diesem Beispiel werden die folgenden Funktionsmöglichkeiten demonstriert:

- Erstellen Sie Modelle für Ereignisdetektoren.
- Erstellen Sie Eingaben.
- Eingaben in ein Detektormodell aufnehmen.
- Evaluieren Sie die Triggerbedingungen.
- Beziehen Sie sich auf Zustandsvariablen in Bedingungen und legen Sie die Werte der Variablen abhängig von den Bedingungen fest.
- Beziehen Sie sich auf Timer in Bedingungen und stellen Sie Timer je nach Bedingungen ein.
- Ergreifen Sie Maßnahmen, die Amazon SNS- und MQTT-Nachrichten senden.

Eingabedefinitionen für ein HVAC-System in AWS IoT Events

A `seedTemperatureInput` wird verwendet, um eine Melderinstanz für einen Bereich zu erstellen und seine Betriebsparameter zu definieren.

Die Konfiguration der Eingänge für HLK-Systeme AWS IoT Events ist wichtig für eine effektive Klimatisierung. Dieses Beispiel zeigt, wie Eingänge eingerichtet werden, die Parameter wie Temperatur, Luftfeuchtigkeit, Belegung und Energieverbrauch erfassen. Erfahren Sie, wie Sie Eingabeattribute definieren, Datenquellen konfigurieren und Vorverarbeitungsregeln einrichten, damit

Ihre Meldermodelle genaue und zeitnahe Informationen für eine optimale Verwaltung und Effizienz erhalten.

Verwendeter CLI-Befehl:

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

Datei: `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" }
    ]
  }
}
```

Antwort:

```
{
    "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."
    }
}
```

Bei Bedarf `temperatureInput` sollte von jedem Sensor in jedem Bereich A gesendet werden.

Verwendeter CLI-Befehl:

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

Datei: `temperatureInput.json`

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

Antwort:

```
{
    "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."
    }
}
```

Definition des Meldermodells für ein HLK-System mit AWS IoT Events

Das `areaDetectorModel` definiert, wie jede Detektorinstanz funktioniert. Jede `state machine` Instanz nimmt Temperatursensormesswerte auf, ändert dann den Status und sendet abhängig von diesen Messwerten Steuermeldungen.

Verwendeter CLI-Befehl:

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

Datei: `areaDetectorModel.json`

```
{
  "detectorModelName": "areaDetectorModel",
  "detectorModelDefinition": {
    "states": [
      {
        "stateName": "start",
        "onEnter": {
          "events": [
            {
              "eventName": "prepare",
              "condition": "true",
              "actions": [
                {
                  "setVariable": {
                    "variableName": "sensorId",
                    "value": "0"
                  }
                },
                {
                  "setVariable": {
                    "variableName": "reportedTemperature",
                    "value": "0.1"
                  }
                },
                {
                  "setVariable": {
                    "variableName": "resetMe",
                    "value": "false"
                  }
                }
              ]
            }
          ]
        },
        "onInput": {
          "transitionEvents": [
            {
              "eventName": "initialize",
```

```
            "condition": "$input.seedTemperatureInput.sensorCount > 0",
            "actions": [
              {
                "setVariable": {
                  "variableName": "rangeHigh",
                  "value": "$input.seedTemperatureInput.rangeHigh"
                }
              },
              {
                "setVariable": {
                  "variableName": "rangeLow",
                  "value": "$input.seedTemperatureInput.rangeLow"
                }
              },
              {
                "setVariable": {
                  "variableName": "desiredTemperature",
                  "value": "$input.seedTemperatureInput.desiredTemperature"
                }
              },
              {
                "setVariable": {
                  "variableName": "averageTemperature",
                  "value": "$input.seedTemperatureInput.desiredTemperature"
                }
              },
              {
                "setVariable": {
                  "variableName": "allowedError",
                  "value": "$input.seedTemperatureInput.allowedError"
                }
              },
              {
                "setVariable": {
                  "variableName": "anomalousHigh",
                  "value": "$input.seedTemperatureInput.anomalousHigh"
                }
              },
              {
                "setVariable": {
                  "variableName": "anomalousLow",
                  "value": "$input.seedTemperatureInput.anomalousLow"
                }
              },
```

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

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


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

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

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

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

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

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

          {
```

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


    {
      "stateName": "cooling",
      "onEnter": {
        "events": [
          {
            "eventName": "delay",
            "condition": "!$variable.noDelay && $variable.enteringNewState",
            "actions": [
              {
                "setTimer": {
                  "timerName": "coolingTimer",
                  "seconds": 180
                }
              },
              {
```

```
                "setVariable": {
                  "variableName": "goodToGo",
                  "value": "false"
                }
              }
            ]
          },
          {
            "eventName": "dontDelay",
            "condition": "$variable.noDelay == true",
            "actions": [
              {
                "setVariable": {
                  "variableName": "goodToGo",
                  "value": "true"
                }
              }
            ]
          },
          {
            "eventName": "beenHere",
            "condition": "true",
            "actions": [
              {
                "setVariable": {
                  "variableName": "enteringNewState",
                  "value": "false"
                }
              }
            ]
          }
        ]
      },

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

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

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

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

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

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

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

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


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

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

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

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

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

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

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

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

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

  ],
```

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

Antwort:

```
{
    "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 Beispiele für ein HVAC-System in AWS IoT Events

In diesem Beispiel `BatchPutMessage` wird verwendet, um eine Melderinstanz für einen Bereich zu erstellen und die anfänglichen Betriebsparameter zu definieren.

Verwendeter CLI-Befehl:

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

Datei: `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}"
```

```
    }
  ]
}
```

Antwort:

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

In diesem Beispiel `BatchPutMessage` wird es verwendet, um Temperatursensorwerte für einen einzelnen Sensor in einem Bereich zu melden.

Verwendeter CLI-Befehl:

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

Datei: `temperatureExample.json`

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

Antwort:

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

In diesem Beispiel `BatchPutMessage` wird verwendet, um die gewünschte Temperatur für einen Bereich zu ändern.

Verwendeter CLI-Befehl:

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

Datei: `seedSetDesiredTemp.json`

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

Antwort:

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

Beispiele für Amazon SNS SNS-Nachrichten, die von der `Area51` Detector-Instance generiert wurden:

```

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"
}
```

In diesem Beispiel verwenden wir die `DescribeDetector` API, um Informationen über den aktuellen Status einer Detector-Instance abzurufen.

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

Antwort:

```
{
    "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 Beispiel für ein HVAC-System in AWS IoT Events

In diesem Beispiel `BatchUpdateDetector` wird verwendet, um Betriebsparameter für eine funktionierende Detektorinstanz zu ändern.

Ein effizientes HVAC-Systemmanagement erfordert häufig Batch-Updates für mehrere Melder. In diesem Abschnitt wird gezeigt, wie die Batch-Aktualisierungsfunktion für Melder verwendet AWS IoT Events wird. Lernen Sie, mehrere Steuerparameter gleichzeitig zu ändern und Schwellenwerte zu aktualisieren, sodass Sie die Reaktionsmaßnahmen für eine Flotte von Geräten anpassen und so Ihre Fähigkeit verbessern können, große Systeme effektiv zu verwalten.

Verwendeter CLI-Befehl:

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

Datei: `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": [
        ]
      }
    }
  ]
}
```

Antwort:

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

Die AWS IoT Core Regel-Engine und AWS IoT Events

Die folgenden Regeln veröffentlichen AWS IoT Events MQTT-Nachrichten erneut als Shadow-Update-Anforderungsnachrichten. Wir gehen davon aus, dass für jeden Bereich, der durch das Detektormodell gesteuert wird, AWS IoT Core Dinge für eine Heiz- und eine Kühleinheit definiert sind.

In diesem Beispiel haben wir Dinge mit dem Namen `Area51HeatingUnit` und definiert`Area51CoolingUnit`.

Verwendeter CLI-Befehl:

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

Datei: `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"
        }
      }
    ]
  }
}
```

Antwort: [leer]

Verwendeter CLI-Befehl:

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

Datei: `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"
        }
      }
    ]
  }
```

```
}
```

Antwort: [leer]

Verwendeter CLI-Befehl:

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

Datei: 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"
        }
      }
    ]
  }
}
```

Antwort: [leer]

Verwendeter CLI-Befehl:

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

Datei: 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"
        }
      }
    ]
  }
}
```

Antwort: [leer]

Beispiel: Ein Kran erkennt Zustände mit AWS IoT Events

Ein Betreiber vieler Krane möchte erkennen, wann die Maschinen gewartet oder ausgetauscht werden müssen, und entsprechende Benachrichtigungen auslösen. Jeder Kran hat einen Motor. Ein Motor sendet Nachrichten (Eingänge) mit Informationen über Druck und Temperatur aus. Der Betreiber benötigt zwei Stufen von Ereignismeldern:

- Ein Ereignismelder auf Kranebene
- Ein Ereignismelder auf Motorebene

Mithilfe von Meldungen von den Motoren (die Metadaten sowohl mit dem `craneId` als auch mit enthaltenmotorid) kann der Bediener die Ereignismelder auf beiden Ebenen mithilfe eines geeigneten Routings ausführen. Wenn die Veranstaltungsbedingungen erfüllt sind, sollten Benachrichtigungen an die entsprechenden Amazon SNS SNS-Themen gesendet werden. Der Betreiber kann die Meldermodelle so konfigurieren, dass keine doppelten Benachrichtigungen ausgelöst werden.

Dieses Beispiel demonstriert die folgenden Funktionsfähigkeiten:

- Eingaben erstellen, lesen, aktualisieren, löschen (CRUD).
- Erstellen, Lesen, Aktualisieren, Löschen (CRUD) von Ereignismeldermodellen und verschiedenen Versionen von Ereignismeldern.

- Weiterleitung eines Eingangs an mehrere Ereignismelder.
- Aufnahme von Eingaben in ein Detektormodell.
- Bewertung von Triggerbedingungen und Lebenszyklusereignissen.
- Fähigkeit, in Bedingungen auf Zustandsvariablen zu verweisen und ihre Werte in Abhängigkeit von den Bedingungen festzulegen.
- Orchestrierung der Laufzeit mit Definition, Status, Trigger-Evaluator und Aktionsausführung.
- Ausführung von Aktionen `ActionsExecutor` mit einem SNS-Ziel.

Senden Sie Befehle als Reaktion auf erkannte Bedingungen in AWS IoT Events

Diese Seite enthält ein Beispiel für die Verwendung von AWS IoT Events Befehlen zum Einrichten von Eingängen, zum Erstellen von Detektormodellen und zum Senden simulierter Sensordaten. Die Beispiele zeigen, wie Industrieanlagen wie Motoren und Krane optimal AWS IoT Events überwacht werden können.

```
#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
```

Ein AWS IoT Events Detektormodell für die Kranüberwachung

Überwachen Sie Ihre Geräte oder Geräteflotten auf Ausfälle oder Betriebsänderungen und lösen Sie bei Auftreten solcher Ereignisse Maßnahmen aus. Sie definieren Meldermodelle in JSON, die Zustände, Regeln und Aktionen spezifizieren. Auf diese Weise können Sie Eingaben wie Temperatur und Druck überwachen, Schwellenwertüberschreitungen verfolgen und Warnmeldungen senden. Die Beispiele zeigen Detektormodelle für einen Kran und Motor, die Überhitzungsprobleme erkennen und über Amazon SNS benachrichtigen, wenn ein Schwellenwert überschritten wird. Sie können Modelle aktualisieren, um das Verhalten zu verfeinern, ohne die Überwachung zu unterbrechen.

Datei: `craneDetectorModel.json`

```
{
    "detectorModelName": "craneDetectorModel",
    "detectorModelDefinition": {
        "states": [
            {
                "stateName": "Running",
                "onEnter": {
                    "events": [
                        {
                            "eventName": "init",
                            "condition": "true",
                            "actions": [
                                {
                                    "setVariable": {
                                        "variableName": "craneThresholdBreached",
                                        "value": "0"
                                    }
                                }
                            ]
                        }
                    ]
                },
                "onInput": {
                    "events": [
                        {
                            "eventName": "Overheated",
                            "condition": "$input.TemperatureInput.temperature > 35",
                            "actions": [
                                {
                                    "setVariable": {
                                        "variableName": "craneThresholdBreached",
                                        "value": "$variable.craneThresholdBreached + 1"
                                    }
                                }
                            ]
                        },
                        {
                            "eventName": "Crane Threshold Breached",
                            "condition": "$variable.craneThresholdBreached > 5",
                            "actions": [
                                {
                                    "sns": {
```

```
                                        "targetArn": "arn:aws:sns:us-
east-1:123456789012:CraneSNSTopic"
                                    }
                                }
                            ]
                        },
                        {
                            "eventName": "Underheated",
                            "condition": "$input.TemperatureInput.temperature < 25",
                            "actions": [
                                {
                                    "setVariable": {
                                        "variableName": "craneThresholdBreached",
                                        "value": "0"
                                    }
                                }
                            ]
                        }
                    ]
                }
            }
        ],
        "initialStateName": "Running"
    },
    "key": "craneid",
    "roleArn": "arn:aws:iam::123456789012:role/columboSNSRole"
}
```

Um ein vorhandenes Meldermodell zu aktualisieren. Datei: `updateCraneDetectorModel.json`

```
{
    "detectorModelName": "craneDetectorModel",
    "detectorModelDefinition": {
        "states": [
            {
                "stateName": "Running",
                "onEnter": {
                    "events": [
                        {
                            "eventName": "init",
                            "condition": "true",
                            "actions": [
                                {
```

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

```
                                    }
                                }
                            ]
                        },
                        {
                            "eventName": "Underheated",
                            "condition": "$input.TemperatureInput.temperature < 10",
                            "actions": [
                                {
                                    "setVariable": {
                                        "variableName": "craneThresholdBreached",
                                        "value": "0"
                                    }
                                }
                            ]
                        }
                    ]
                }
            }
        ],
        "initialStateName": "Running"
    },
    "roleArn": "arn:aws:iam::123456789012:role/columboSNSRole"
}
```

Datei: `motorDetectorModel.json`

```
{
    "detectorModelName": "motorDetectorModel",
    "detectorModelDefinition": {
        "states": [
            {
                "stateName": "Running",
                "onEnter": {
                    "events": [
                        {
                            "eventName": "init",
                            "condition": "true",
                            "actions": [
                                {
                                    "setVariable": {
                                        "variableName": "motorThresholdBreached",
                                        "value": "0"
```

```
                                    }
                                }
                            ]
                        }
                    ]
                },
                "onInput": {
                    "events": [
                        {
                            "eventName": "Overheated And Overpressurized",
                            "condition": "$input.PressureInput.pressure > 70 &&
 $input.TemperatureInput.temperature > 30",
                            "actions": [
                                {
                                    "setVariable": {
                                        "variableName": "motorThresholdBreached",
                                        "value": "$variable.motorThresholdBreached + 1"
                                    }
                                }
                            ]
                        },
                        {
                            "eventName": "Motor Threshold Breached",
                            "condition": "$variable.motorThresholdBreached > 5",
                            "actions": [
                                {
                                    "sns": {
                                        "targetArn": "arn:aws:sns:us-
east-1:123456789012:MotorSNSTopic"
                                    }
                                }
                            ]
                        }
                    ]
                }
            }
        ],
        "initialStateName": "Running"
    },
    "key": "motorId",
    "roleArn": "arn:aws:iam::123456789012:role/columboSNSRole"
}
```

Um ein vorhandenes Meldermodell zu aktualisieren. Datei: `updateMotorDetectorModel.json`

```
{
    "detectorModelName": "motorDetectorModel",
    "detectorModelDefinition": {
        "states": [
            {
                "stateName": "Running",
                "onEnter": {
                    "events": [
                        {
                            "eventName": "init",
                            "condition": "true",
                            "actions": [
                                {
                                    "setVariable": {
                                        "variableName": "motorThresholdBreached",
                                        "value": "0"
                                    }
                                }
                            ]
                        }
                    ]
                },
                "onInput": {
                    "events": [
                        {
                            "eventName": "Overheated And Overpressurized",
                            "condition": "$input.PressureInput.pressure > 70 &&
 $input.TemperatureInput.temperature > 30",
                            "actions": [
                                {
                                    "setVariable": {
                                        "variableName": "motorThresholdBreached",
                                        "value": "$variable.motorThresholdBreached + 1"
                                    }
                                }
                            ]
                        },
                        {
                            "eventName": "Motor Threshold Breached",
                            "condition": "$variable.motorThresholdBreached > 5",
                            "actions": [
                                {
```

```
                                    "sns": {
                                        "targetArn": "arn:aws:sns:us-
east-1:123456789012:MotorSNSTopic"
                                    }
                                }
                            ]
                        }
                    ]
                }
            }
        ],
        "initialStateName": "Running"
    },
    "roleArn": "arn:aws:iam::123456789012:role/columboSNSRole"
}
```

AWS IoT Events Eingänge für die Kranüberwachung

In diesem Beispiel zeigen wir, wie man Eingänge für ein Kranüberwachungssystem einrichtet mit AWS IoT Events. Es erfasst Druck- und Temperatureingaben, um zu veranschaulichen, wie Eingaben für die Überwachung komplexer Industrieanlagen strukturiert werden können.

Datei: `pressureInput.json`

```
{
    "inputName": "PressureInput",
    "inputDescription": "this is a pressure input description",
    "inputDefinition": {
        "attributes": [
          {"jsonPath": "pressure"}
        ]
    }
}
```

Datei: `temperatureInput.json`

```
{
    "inputName": "TemperatureInput",
    "inputDescription": "this is temperature input description",
    "inputDefinition": {
        "attributes": [
```

```
            {"jsonPath": "temperature"}
        ]
    }
}
```

Senden Sie Alarm- und Betriebsmeldungen mit AWS IoT Events

Eine effektive Nachrichtenverarbeitung ist bei Kranüberwachungssystemen wichtig. In diesem Abschnitt wird gezeigt, wie die Konfiguration AWS IoT Events für die Verarbeitung und Beantwortung verschiedener Meldungstypen von Kransensoren konfiguriert wird. Das Einrichten von Alarmen auf der Grundlage einer bestimmten Nachricht kann Ihnen dabei helfen, Statusaktualisierungen zu analysieren, zu filtern und weiterzuleiten, um entsprechende Aktionen auszulösen.

Datei: `highPressureMessage.json`

```
{
   "messages": [
        {
           "messageId": "1",
           "inputName": "PressureInput",
           "payload": "{\"craneid\": \"100009\", \"pressure\": 80, \"motorid\":
 \"200009\"}"

        }
    ]
}
```

Datei: `highTemperatureMessage.json`

```
{
   "messages": [
        {
           "messageId": "2",
           "inputName": "TemperatureInput",
           "payload": "{\"craneid\": \"100009\", \"temperature\": 40, \"motorid\":
 \"200009\"}"
        }
    ]
}
```

Datei: `lowPressureMessage.json`

```
{
   "messages": [
        {
           "messageId": "1",
           "inputName": "PressureInput",
           "payload": "{\"craneid\": \"100009\", \"pressure\": 20, \"motorid\":
 \"200009\"}"
        }
    ]
}
```

Datei: `lowTemperatureMessage.json`

```
{
   "messages": [
        {
           "messageId": "2",
           "inputName": "TemperatureInput",
           "payload": "{\"craneid\": \"100009\", \"temperature\": 20, \"motorid\":
 \"200009\"}"
        }
    ]
}
```

Beispiel: AWS IoT Events Ereigniserkennung mit Sensoren und Anwendungen

Dieses Detektormodell ist eine der Vorlagen, die auf der AWS IoT Events Konsole verfügbar sind. Es ist hier der Einfachheit halber enthalten.

Dieses Beispiel zeigt AWS IoT Events die Anwendungsereigniserkennung anhand von Sensordaten. Es zeigt, wie Sie ein Detektormodell erstellen können, das bestimmte Ereignisse überwacht, sodass Sie entsprechende Aktionen auslösen können. Sie können mehrere Sensoreingänge erstellen, komplexe Ereignisbedingungen definieren und abgestufte Reaktionsmechanismen einrichten.

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

```
                "onInput": {
                    "transitionEvents": [],
                    "events": []
                },
                "stateName": "Device_exception",
                "onEnter": {
                    "events": [
                        {
                            "eventName": "Send_mqtt",
                            "actions": [
                                {
                                    "iotTopicPublish": {
                                        "mqttTopic": "Device_stolen"
                                    }
                                }
                            ],
                            "condition": "true"
                        }
                    ]
                },
                "onExit": {
                    "events": []
                }
            },
            {
                "onInput": {
                    "transitionEvents": [
                        {
                            "eventName": "To_in_use",
                            "actions": [],
                            "condition": "$variable.position !=
$input.AWS_IoTEvents_Blueprints_Tracking_DeviceInput.gps_position",
                            "nextState": "Device_in_use"
                        }
                    ],
                    "events": []
                },
                "stateName": "Device_idle",
                "onEnter": {
                    "events": [
                        {
                            "eventName": "Set_position",
                            "actions": [
                                {
```

```
                                    "setVariable": {
                                        "variableName": "position",
                                        "value":
 "$input.AWS_IoTEvents_Blueprints_Tracking_DeviceInput.gps_position"
                                    }
                                }
                            ],
                            "condition": "true"
                        }
                    ]
                },
                "onExit": {
                    "events": []
                }
            },
            {
                "onInput": {
                    "transitionEvents": [
                        {
                            "eventName": "To_exception",
                            "actions": [],
                            "condition":
 "$input.AWS_IoTEvents_Blueprints_Tracking_UserInput.device_id !=
 $input.AWS_IoTEvents_Blueprints_Tracking_DeviceInput.device_id",
                            "nextState": "Device_exception"
                        }
                    ],
                    "events": []
                },
                "stateName": "Device_in_use",
                "onEnter": {
                    "events": []
                },
                "onExit": {
                    "events": []
                }
            }
        ],
        "initialStateName": "Device_idle"
    }
}
```

Beispiel: Gerät HeartBeat zur Überwachung von Geräteverbindungen mit AWS IoT Events

Dieses Meldermodell ist eine der Vorlagen, die auf der AWS IoT Events Konsole verfügbar sind. Es ist hier der Einfachheit halber enthalten.

Das Beispiel Defective Heart Beat (DHB) veranschaulicht, wie es bei der Überwachung im Gesundheitswesen eingesetzt werden AWS IoT Events kann. Dieses Beispiel zeigt, wie Sie ein Detektormodell erstellen können, das Herzfrequenzdaten analysiert, unregelmäßige Muster erkennt und entsprechende Reaktionen auslöst. Erfahren Sie, wie Sie Eingaben einrichten, Schwellenwerte definieren und Warnmeldungen für potenzielle Herzprobleme konfigurieren. Dies zeigt AWS IoT Events die Vielseitigkeit bei verwandten Anwendungen im Gesundheitswesen.

```
{
    "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"
    }
}
```

Beispiel: Ein ISA-Alarm in AWS IoT Events

Dieses Meldermodell ist eine der Vorlagen, die in der AWS IoT Events Konsole verfügbar sind. Es ist hier der Einfachheit halber enthalten.

```
{
    "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"
}
```

Beispiel: Erstellen Sie einen einfachen Alarm mit AWS IoT Events

Dieses Meldermodell ist eine der Vorlagen, die auf der AWS IoT Events Konsole verfügbar sind. Es ist hier der Einfachheit halber enthalten.

```
{
    "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"
}
```

Überwachung mit Alarmen in AWS IoT Events

AWS IoT Events Mithilfe von Alarmen können Sie Ihre Daten auf Änderungen überwachen. Bei den Daten kann es sich um Kennzahlen handeln, die Sie für Ihre Ausrüstung und Prozesse messen. Sie können Alarme erstellen, die Benachrichtigungen senden, wenn ein Schwellenwert überschritten wird. Alarme helfen Ihnen dabei, Probleme zu erkennen, die Wartung zu rationalisieren und die Leistung Ihrer Geräte und Prozesse zu optimieren.

Alarme sind Beispiele von Alarmmodellen. Das Alarmmodell legt fest, was erkannt werden soll, wann Benachrichtigungen gesendet werden sollen, wer benachrichtigt wird und vieles mehr. Sie können auch eine oder mehrere unterstützte Aktionen angeben, die ausgeführt werden, wenn sich der Alarmstatus ändert. AWS IoT Events leitet aus Ihren Daten abgeleitete Eingabeattribute an die entsprechenden Alarme weiter. Wenn die Daten, die Sie überwachen, außerhalb des angegebenen Bereichs liegen, wird der Alarm ausgelöst. Sie können die Alarme auch bestätigen oder sie in den Schlummermodus versetzen.

Arbeitet mit AWS IoT SiteWise

Sie können AWS IoT Events Alarme verwenden, um die Eigenschaften von Vermögenswerten in zu überwachen AWS IoT SiteWise. AWS IoT SiteWise sendet Eigenschaftswerte von Vermögenswerten an AWS IoT Events Alarme. AWS IoT Events sendet den Alarmstatus an AWS IoT SiteWise.

AWS IoT SiteWise unterstützt auch externe Alarme. Sie können externe Alarme wählen, wenn Sie Alarme außerhalb von verwenden AWS IoT SiteWise und über eine Lösung verfügen, die Alarmstatusdaten zurückgibt. Der externe Alarm enthält eine Messeigenschaft, die die Alarmzustandsdaten aufnimmt.

AWS IoT SiteWise wertet den Status externer Alarme nicht aus. Außerdem können Sie einen externen Alarm nicht bestätigen oder deaktivieren, wenn sich der Alarmstatus ändert.

Sie können die SiteWise Überwachungsfunktion verwenden, um den Status externer Alarme in SiteWise Monitor-Portalen einzusehen.

Weitere Informationen finden Sie unter Überwachen von Daten mit Alarmen im AWS IoT SiteWise Benutzerhandbuch und Überwachen mit Alarmen im SiteWise Monitor-Anwendungshandbuch.

Bestätigen Sie den Ablauf

Wenn Sie ein Alarmmodell erstellen, wählen Sie aus, ob der Bestätigungsfluss aktiviert werden soll. Wenn Sie den Bestätigungsfluss aktivieren, wird Ihr Team benachrichtigt, wenn sich der Alarmstatus ändert. Ihr Team kann den Alarm bestätigen und eine Notiz hinterlassen. Sie können beispielsweise die Informationen zum Alarm und die Maßnahmen, die Sie ergreifen werden, um das Problem zu beheben, angeben. Wenn die Daten, die Sie überwachen, außerhalb des angegebenen Bereichs liegen, wird der Alarm ausgelöst.

Alarme haben die folgenden Status:

`DISABLED`

Wenn sich der Alarm im `DISABLED` Status befindet, ist er nicht bereit, Daten auszuwerten. Um den Alarm zu aktivieren, müssen Sie den Alarm in den `NORMAL` Status ändern.

`NORMAL`

Wenn sich der Alarm im `NORMAL` Status befindet, ist er bereit, Daten auszuwerten.

`ACTIVE`

Befindet sich der Alarm im `ACTIVE` Status, wird der Alarm ausgelöst. Die Daten, die Sie überwachen, liegen außerhalb des angegebenen Bereichs.

`ACKNOWLEDGED`

Wenn sich der Alarm im `ACKNOWLEDGED` Status befindet, wurde der Alarm ausgelöst und Sie haben den Alarm bestätigt.

`LATCHED`

Der Alarm wurde ausgelöst, aber Sie haben den Alarm nach einer gewissen Zeit nicht bestätigt. Der Alarm wechselt automatisch in den `NORMAL` Status.

`SNOOZE_DISABLED`

Wenn sich der Alarm im `SNOOZE_DISABLED` Status befindet, ist der Alarm für einen bestimmten Zeitraum deaktiviert. Nach Ablauf der Schlummerzeit wechselt der Alarm automatisch in den `NORMAL` Status.

Erstellen eines Alarmmodells in AWS IoT Events

Sie können AWS IoT Events Alarme verwenden, um Ihre Daten zu überwachen und sich benachrichtigen zu lassen, wenn ein Schwellenwert überschritten wird. Alarme stellen Parameter bereit, die Sie verwenden, um ein Alarmmodell zu erstellen oder zu konfigurieren. Sie können die AWS IoT Events Konsole oder AWS IoT Events API verwenden, um das Alarmmodell zu erstellen oder zu konfigurieren. Wenn Sie das Alarmmodell konfigurieren, werden Änderungen wirksam, sobald neue Daten eintreffen.

Voraussetzungen

Die folgenden Anforderungen gelten, wenn Sie ein Alarmmodell erstellen.

- Sie können ein Alarmmodell erstellen, um ein Eingabeattribut AWS IoT Events oder eine Anlageneigenschaft in zu überwachen AWS IoT SiteWise.
  - Wenn Sie ein Eingabeattribut in überwachen möchten AWS IoT Events, Erstellen Sie eine Eingabe für Modelle in AWS IoT Events bevor Sie das Alarmmodell erstellen.
  - Wenn Sie sich dafür entscheiden, eine Anlageneigenschaft zu überwachen, müssen Sie AWS IoT SiteWise vor der Erstellung des Alarmmodells ein Asset-Modell in erstellen.
- Sie müssen über eine IAM-Rolle verfügen, die es Ihrem Alarm ermöglicht, Aktionen auszuführen und auf AWS Ressourcen zuzugreifen. Weitere Informationen finden Sie unter Berechtigungen einrichten für AWS IoT Events.
- Alle AWS Ressourcen, die in diesem Tutorial verwendet werden, müssen sich in derselben AWS Region befinden.

Ein Alarmmodell (Konsole) erstellen

Im Folgenden wird gezeigt, wie Sie ein Alarmmodell zur Überwachung eines AWS IoT Events Attributs in der AWS IoT Events Konsole erstellen.

1. Melden Sie sich bei der AWS IoT Events -Konsole an.
2. Wählen Sie im Navigationsbereich die Option Alarmmodelle aus.
3. Wählen Sie auf der Seite Alarmmodelle die Option Alarmmodell erstellen aus.
4. Gehen Sie im Abschnitt Details zum Alarmmodell wie folgt vor:

   a. Geben Sie einen eindeutigen Namen ein.

b. (Optional) Geben Sie eine Beschreibung ein.

5. Gehen Sie im Abschnitt Alarmziel wie folgt vor:

> **Important**
>
> Wenn Sie sich für eine AWS IoT SiteWise Anlageneigenschaft entscheiden, müssen Sie in ein Asset-Modell erstellt haben AWS IoT SiteWise.

a. Wählen Sie das AWS IoT Events Eingabeattribut.

b. Wählen Sie die Eingabe aus.

c. Wählen Sie den Eingabeattributschlüssel aus. Dieses Eingabeattribut wird als Schlüssel zum Erstellen des Alarms verwendet. AWS IoT Events leitet die mit diesem Schlüssel verknüpften Eingaben an den Alarm weiter.

> **Important**
>
> Wenn die Nutzlast der Eingabenachricht diesen Eingabeattributschlüssel nicht enthält oder wenn sich der Schlüssel nicht in demselben JSON-Pfad befindet, der im Schlüssel angegeben ist, schlägt die Eingabe in der Nachricht fehl. AWS IoT Events

6. Im Abschnitt Schwellenwertdefinitionen definieren Sie das Eingabeattribut, den Schwellenwert und den Vergleichsoperator, AWS IoT Events mit dem der Status des Alarms geändert wird.

a. Wählen Sie unter Eingabeattribut das Attribut aus, das Sie überwachen möchten.

Jedes Mal, wenn dieses Eingabeattribut neue Daten empfängt, werden diese ausgewertet, um den Status des Alarms zu bestimmen.

b. Wählen Sie unter Operator den Vergleichsoperator aus. Der Operator vergleicht Ihr Eingabeattribut mit dem Schwellenwert für Ihr Attribut.

Sie können aus diesen Optionen wählen:

- > größer als
- >= größer als oder gleich
- < kleiner als
- <= kleiner als oder gleich

- = gleich
- ! = nicht gleich

c. Geben Sie für den Schwellenwert eine Zahl ein, oder wählen Sie ein Attribut in den AWS IoT Events Eingaben aus. AWS IoT Events vergleicht diesen Wert mit dem Wert des von Ihnen ausgewählten Eingabeattributs.

d. (Optional) Verwenden Sie für Schweregrad eine Zahl, von der Ihr Team weiß, dass sie den Schweregrad dieses Alarms wiedergibt.

7. (Optional) Konfigurieren Sie im Abschnitt Benachrichtigungseinstellungen die Benachrichtigungseinstellungen für den Alarm.

   Sie können bis zu 10 Benachrichtigungen hinzufügen. Gehen Sie für Benachrichtigung 1 wie folgt vor:

   a. Wählen Sie für Protokoll eine der folgenden Optionen aus:

      - E-Mail und Text — Der Alarm sendet eine SMS-Benachrichtigung und eine E-Mail-Benachrichtigung.
      - E-Mail — Der Alarm sendet eine E-Mail-Benachrichtigung.
      - Text — Der Alarm sendet eine SMS-Benachrichtigung.

   b. Geben Sie als Absender die E-Mail-Adresse an, an die Benachrichtigungen zu diesem Alarm gesendet werden können.

      Um Ihrer Absenderliste weitere E-Mail-Adressen hinzuzufügen, wählen Sie Absender hinzufügen.

   c. (Optional) Wählen Sie unter Empfänger den Empfänger aus.

      Um Ihrer Empfängerliste weitere Benutzer hinzuzufügen, wählen Sie Neuen Benutzer hinzufügen. Sie müssen Ihrem IAM Identity Center-Shop neue Benutzer hinzufügen, bevor Sie sie zu Ihrem Alarmmodell hinzufügen können. Weitere Informationen finden Sie unter Verwalten Sie den Zugriff von Alarmempfängern auf das IAM Identity Center in AWS IoT Events.

   d. (Optional) Geben Sie unter Zusätzliche benutzerdefinierte Nachricht eine Nachricht ein, die beschreibt, was der Alarm erkennt und welche Maßnahmen die Empfänger ergreifen sollten.

8. Im Abschnitt Instanz können Sie alle Alarminstanzen aktivieren oder deaktivieren, die auf der Grundlage dieses Alarmmodells erstellt wurden.

9. Gehen Sie im Abschnitt Erweiterte Einstellungen wie folgt vor:

a. Für den Acknowledge-Flow können Sie Benachrichtigungen aktivieren oder deaktivieren.

   - Wenn Sie Aktiviert wählen, erhalten Sie eine Benachrichtigung, wenn sich der Alarmstatus ändert. Sie müssen die Benachrichtigung bestätigen, bevor der Alarmstatus wieder normal werden kann.
   - Wenn Sie Deaktiviert wählen, ist keine Aktion erforderlich. Der Alarm wechselt automatisch in den Normalzustand, wenn die Messung in den angegebenen Bereich zurückkehrt.

   Weitere Informationen finden Sie unter Bestätigen Sie den Ablauf.

b. Wählen Sie für Berechtigungen eine der folgenden Optionen aus:

   - Sie können eine neue Rolle anhand von AWS Richtlinienvorlagen erstellen und AWS IoT Events automatisch eine IAM-Rolle für Sie erstellen.
   - Sie können eine vorhandene IAM-Rolle verwenden, die es diesem Alarmmodell ermöglicht, Aktionen auszuführen und auf andere AWS Ressourcen zuzugreifen.

   Weitere Informationen finden Sie unter Identitäts- und Zugriffsverwaltung für AWS IoT Events.

c. Für zusätzliche Benachrichtigungseinstellungen können Sie Ihre AWS Lambda Funktion zur Verwaltung von Alarmbenachrichtigungen bearbeiten. Wählen Sie eine der folgenden Optionen für Ihre AWS Lambda Funktion:

   - Neue AWS Lambda Funktion erstellen — AWS IoT Events erstellt eine neue AWS Lambda Funktion für Sie.
   - Eine bestehende AWS Lambda Funktion verwenden — Verwenden Sie eine bestehende AWS Lambda Funktion, indem Sie einen AWS Lambda Funktionsnamen wählen.

   Weitere Hinweise zu den möglichen Aktionen finden Sie unterAWS IoT Events mit anderen AWS Diensten arbeiten.

d. (Optional) Für die Aktion Status festlegen können Sie eine oder mehrere AWS IoT Events Aktionen hinzufügen, die ausgeführt werden sollen, wenn sich der Alarmstatus ändert.

10. (Optional) Sie können Tags hinzufügen, um Ihre Alarme zu verwalten. Weitere Informationen finden Sie unter Markieren Ihrer AWS IoT Events -Ressourcen.

11. Wählen Sie Erstellen aus.

Reagieren auf Alarme in AWS IoT Events

Die effektive Reaktion auf Alarme ist ein wichtiger Aspekt der Verwaltung von IoT-Systemen mit AWS IoT Events. Erkunden Sie verschiedene Möglichkeiten zur Konfiguration und Handhabung von Alarmen, darunter die Einrichtung von Benachrichtigungskanälen, die Definition von Eskalationsverfahren und die Implementierung automatisierter Reaktionsmaßnahmen. Erfahren Sie, wie Sie differenzierte Alarmbedingungen erstellen, Warnmeldungen priorisieren und andere AWS Dienste integrieren, um ein reaktionsschnelles Alarmmanagementsystem für Ihre IoT-Anwendungen aufzubauen.

Wenn Sie den Bestätigungsfluss aktiviert haben, erhalten Sie Benachrichtigungen, wenn sich der Alarmstatus ändert. Um auf den Alarm zu reagieren, können Sie den Alarm bestätigen, deaktivieren, aktivieren, zurücksetzen oder die Schlummerfunktion aktivieren.

Console

Im Folgenden wird gezeigt, wie Sie auf einen Alarm in der AWS IoT Events Konsole reagieren.

1. Melden Sie sich bei der AWS IoT Events -Konsole an.
2. Wählen Sie im Navigationsbereich die Option Alarmmodelle aus.
3. Wählen Sie das Zielalarmmodell aus.
4. Wählen Sie im Bereich Liste der Alarme den Zielalarm aus.
5. Sie können unter Aktionen eine der folgenden Optionen wählen:

   - Bestätigen — Der Alarm wechselt in den `ACKNOWLEDGED` Status.
   - Deaktivieren — Der Alarm wechselt in den `DISABLED` Status.
   - Aktivieren — Der Alarm wechselt in den `NORMAL` Status.
   - Reset — Der Alarm wechselt in den `NORMAL` Status.
   - Schalten Sie die Schlummerfunktion ein und gehen Sie dann wie folgt vor:

     1. Wählen Sie die Schlummerlänge oder geben Sie eine benutzerdefinierte Schlummerlänge ein.
     2. Wählen Sie Speichern.

     Der Alarm wechselt in den Status `SNOOZE_DISABLED`

Weitere Informationen zu den Alarmzuständen finden Sie unterBestätigen Sie den Ablauf.

API

Um auf einen oder mehrere Alarme zu reagieren, können Sie die folgenden AWS IoT Events API-Operationen verwenden:

- BatchAcknowledgeAlarm
- BatchDisableAlarm
- BatchEnableAlarm
- BatchResetAlarm
- BatchSnoozeAlarm

Verwaltung von Alarmbenachrichtigungen in AWS IoT Events

AWS IoT Events lässt sich in Lambda integrieren und bietet benutzerdefinierte Funktionen zur Ereignisverarbeitung. In diesem Abschnitt wird untersucht, wie Sie Lambda-Funktionen in Ihren AWS IoT Events Detektormodellen verwenden, sodass Sie komplexe Logik ausführen, mit externen Diensten interagieren und eine ausgefeilte Ereignisbehandlung implementieren können.

AWS IoT Events verwendet eine Lambda-Funktion zur Verwaltung von Alarmbenachrichtigungen. Sie können die von bereitgestellte Lambda-Funktion verwenden AWS IoT Events oder eine neue erstellen.

Themen

- Erstellen einer Lambda-Funktion in AWS IoT Events
- Verwenden der Lambda-Funktion von AWS IoT Events
- Verwalten Sie den Zugriff von Alarmempfängern auf das IAM Identity Center in AWS IoT Events

Erstellen einer Lambda-Funktion in AWS IoT Events

AWS IoT Events bietet eine Lambda-Funktion, die es Alarmen ermöglicht, E-Mail- und SMS-Benachrichtigungen zu senden und zu empfangen.

Voraussetzungen

Die folgenden Anforderungen gelten, wenn Sie eine Lambda-Funktion für Alarme erstellen:

- Wenn Ihr Alarm SMS-Benachrichtigungen sendet, stellen Sie sicher, dass Amazon SNS für die Zustellung von SMS-Nachrichten konfiguriert ist.
  - Weitere Informationen finden Sie in der folgenden Dokumentation:
    - Mobiles Textnachrichten mit Amazon SNS und Originationsidentitäten für Amazon SNS SNS-SMS-Nachrichten im Amazon Simple Notification Service Developer Guide.
    - Was ist AWS End User Messaging SMS? im AWS SMS Benutzerhandbuch.
- Wenn Ihr Alarm E-Mail- oder SMS-Benachrichtigungen sendet, benötigen Sie eine IAM-Rolle, die die Arbeit mit Amazon SES und Amazon SNS ermöglicht AWS Lambda .

  Beispiel für eine Richtlinie:

  JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ses:GetIdentityVerificationAttributes",
                "ses:SendEmail",
                "ses:VerifyEmailIdentity"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "sns:Publish",
                "sns:OptInPhoneNumber",
                "sns:CheckIfPhoneNumberIsOptedOut",
                "sms-voice:DescribeOptedOutNumbers"
            ],
            "Resource": "*"
        },
        {
```

```
            "Effect": "Deny",
            "Action": "sns:Publish",
            "Resource": "arn:aws:sns:*:*:*"
        },
        {
          "Effect" : "Allow",
          "Action" : [
            "logs:CreateLogGroup",
            "logs:CreateLogStream",
            "logs:PutLogEvents"
          ],
          "Resource" : "*"
        }
    ]
}
```

- Sie müssen dieselbe AWS Region für AWS IoT Events sowohl als auch wählen AWS Lambda. Eine Liste der unterstützten Regionen finden Sie unter AWS IoT Events Endpunkte und Kontingente und AWS Lambda Endpunkte und Kontingente in der. Allgemeine Amazon Web Services-Referenz

Stellen Sie eine Lambda-Funktion für AWS IoT Events die Verwendung bereit CloudFormation

In diesem Tutorial wird eine CloudFormation Vorlage verwendet, um eine Lambda-Funktion bereitzustellen. Diese Vorlage erstellt automatisch eine IAM-Rolle, die es der Lambda-Funktion ermöglicht, mit Amazon SES und Amazon SNS zu arbeiten.

Im Folgenden wird gezeigt, wie Sie AWS Command Line Interface (AWS CLI) verwenden, um einen Stack zu erstellen. CloudFormation

1. Führen Sie im Terminal Ihres Geräts den Befehl aus, `aws --version` um zu überprüfen, ob Sie den installiert haben AWS CLI. Weitere Informationen finden Sie unter Installieren oder Aktualisieren auf die neueste Version der AWS CLI im Benutzerhandbuch für AWS Command Line Interface .
2. Führen Sie den Vorgang aus`aws configure list`, um zu überprüfen, ob Sie das AWS CLI in der AWS Region konfiguriert haben, die alle Ihre AWS Ressourcen für dieses Tutorial enthält. Weitere Informationen finden Sie im AWS Command Line Interface Benutzerhandbuch unter Einrichten und Anzeigen von Konfigurationseinstellungen mithilfe von Befehlen

3. Laden Sie die CloudFormation Vorlage NotificationLambda.Template.Yaml.zip herunter.

   > **Note**
   >
   > Wenn Sie Schwierigkeiten beim Herunterladen der Datei haben, finden Sie die Vorlage auch im. CloudFormation Vorlage

4. Entpacken Sie den Inhalt und speichern Sie die Datei lokal als `notificationLambda.template.yaml`.
5. Öffnen Sie ein Terminal auf Ihrem Gerät und navigieren Sie zu dem Verzeichnis, in das Sie die `notificationLambda.template.yaml` Datei heruntergeladen haben.
6. Führen Sie den folgenden Befehl aus, um einen CloudFormation Stack zu erstellen:

   ```
   aws cloudformation create-stack --stack-name notificationLambda-stack --template-
   body file://notificationLambda.template.yaml --capabilities CAPABILITY_IAM
   ```

Sie können diese CloudFormation Vorlage ändern, um die Lambda-Funktion und ihr Verhalten anzupassen.

> **Note**
>
> AWS Lambda wiederholt Funktionsfehler zweimal. Wenn die Kapazität der Funktion nicht für die Verarbeitung aller eingehenden Anforderungen ausreicht, verbleiben die an die Funktion zu sendenden Ereignisse möglicherweise stunden- oder tagelang in der Warteschlange. Sie können für die Funktion eine Warteschlange für unzugestellte Nachrichten (DLQ) konfigurieren, um Ereignisse aufzuzeichnen, die nicht erfolgreich verarbeitet wurden. Weitere Informationen finden Sie unter Asynchroner Aufruf im AWS Lambda Entwicklerhandbuch.

Sie können den Stack auch in der Konsole erstellen oder konfigurieren. CloudFormation Weitere Informationen finden Sie im AWS CloudFormation Benutzerhandbuch unter Arbeiten mit Stacks.

Erstellen einer benutzerdefinierten Lambda-Funktion für AWS IoT Events

Sie können eine Lambda-Funktion erstellen oder die von AWS IoT Events bereitgestellte ändern.

Die folgenden Anforderungen gelten, wenn Sie eine benutzerdefinierte Lambda-Funktion erstellen.

- Fügen Sie Berechtigungen hinzu, die es Ihrer Lambda-Funktion ermöglichen, bestimmte Aktionen auszuführen und auf AWS Ressourcen zuzugreifen.
- Wenn Sie die von bereitgestellte Lambda-Funktion verwenden AWS IoT Events, stellen Sie sicher, dass Sie die Python 3.7-Laufzeit wählen.

Beispiel für eine Lambda-Funktion:

```
import boto3
import json
import logging
import datetime
logger = logging.getLogger()
logger.setLevel(logging.INFO)
ses = boto3.client('ses')
sns = boto3.client('sns')
def check_value(target):
  if target:
    return True
  return False

# Check whether email is verified. Only verified emails are allowed to send emails to
 or from.
def check_email(email):
  if not check_value(email):
    return False
  result = ses.get_identity_verification_attributes(Identities=[email])
  attr = result['VerificationAttributes']
  if (email not in attr or attr[email]['VerificationStatus'] != 'Success'):
      logging.info('Verification email for {} sent. You must have all the emails
 verified before sending email.'.format(email))
      ses.verify_email_identity(EmailAddress=email)
      return False
  return True

# Check whether the phone holder has opted out of receiving SMS messages from your
 account
def check_phone_number(phone_number):
  try:
    result = sns.check_if_phone_number_is_opted_out(phoneNumber=phone_number)
    if (result['isOptedOut']):
        logger.info('phoneNumber {} is not opt in of receiving SMS messages. Phone
 number must be opt in first.'.format(phone_number))
```

```
        return False
    return True
  except Exception as e:
    logging.error('Your phone number {} must be in E.164 format in SSO. Exception
 thrown: {}'.format(phone_number, e))
    return False

def check_emails(emails):
  result = True
  for email in emails:
      if not check_email(email):
          result = False
  return result

def lambda_handler(event, context):
  logging.info('Received event: ' + json.dumps(event))
  nep = json.loads(event.get('notificationEventPayload'))
  alarm_state = nep['alarmState']
  default_msg = 'Alarm ' + alarm_state['stateName'] + '\n'
  timestamp =
 datetime.datetime.utcfromtimestamp(float(nep['stateUpdateTime'])/1000).strftime('%Y-
%m-%d %H:%M:%S')
  alarm_msg = "{} {} {} at {} UTC ".format(nep['alarmModelName'], nep.get('keyValue',
 'Singleton'), alarm_state['stateName'], timestamp)
  default_msg += 'Sev: ' + str(nep['severity']) + '\n'
  if (alarm_state['ruleEvaluation']):
    property = alarm_state['ruleEvaluation']['simpleRule']['inputProperty']
    default_msg += 'Current Value: ' + str(property) + '\n'
    operator = alarm_state['ruleEvaluation']['simpleRule']['operator']
    threshold = alarm_state['ruleEvaluation']['simpleRule']['threshold']
    alarm_msg += '({} {} {})'.format(str(property), operator, str(threshold))
  default_msg += alarm_msg + '\n'

  emails = event.get('emailConfigurations', [])
  logger.info('Start Sending Emails')
  for email in emails:
    from_adr = email.get('from')
    to_adrs = email.get('to', [])
    cc_adrs = email.get('cc', [])
    bcc_adrs = email.get('bcc', [])
    msg = default_msg + '\n' + email.get('additionalMessage', '')
    subject = email.get('subject', alarm_msg)
    fa_ver = check_email(from_adr)
    tas_ver = check_emails(to_adrs)
```

```
    ccas_ver = check_emails(cc_adrs)
    bccas_ver = check_emails(bcc_adrs)
    if (fa_ver and tas_ver and ccas_ver and bccas_ver):
      ses.send_email(Source=from_adr,
                     Destination={'ToAddresses': to_adrs, 'CcAddresses': cc_adrs,
'BccAddresses': bcc_adrs},
                     Message={'Subject': {'Data': subject}, 'Body': {'Text': {'Data':
msg}}})
      logger.info('Emails have been sent')

  logger.info('Start Sending SNS message to SMS')
  sns_configs = event.get('smsConfigurations', [])
  for sns_config in sns_configs:
    sns_msg = default_msg + '\n' + sns_config.get('additionalMessage', '')
    phone_numbers = sns_config.get('phoneNumbers', [])
    sender_id = sns_config.get('senderId')
    for phone_number in phone_numbers:
        if check_phone_number(phone_number):
          if check_value(sender_id):
            sns.publish(PhoneNumber=phone_number, Message=sns_msg,
MessageAttributes={'AWS.SNS.SMS.SenderID':{'DataType': 'String','StringValue':
sender_id}})
          else:
            sns.publish(PhoneNumber=phone_number, Message=sns_msg)
          logger.info('SNS messages have been sent')
```

Weitere Informationen finden Sie unter Was ist AWS Lambda? im AWS Lambda - Entwicklerhandbuch.

CloudFormation Vorlage

Verwenden Sie die folgende CloudFormation Vorlage, um Ihre Lambda-Funktion zu erstellen.

```
AWSTemplateFormatVersion: '2010-09-09'
Description: 'Notification Lambda for Alarm Model'
Resources:
  NotificationLambdaRole:
    Type: AWS::IAM::Role
    Properties:
      AssumeRolePolicyDocument:
        Statement:
          - Effect: Allow
            Principal:
```

```
            Service: lambda.amazonaws.com
          Action: sts:AssumeRole
    Path: "/"
    ManagedPolicyArns:
      - 'arn:aws:iam::aws:policy/AWSLambdaExecute'
    Policies:
      - PolicyName: "NotificationLambda"
        PolicyDocument:
          Version: "2012-10-17"
          Statement:
            - Effect: "Allow"
              Action:
                - "ses:GetIdentityVerificationAttributes"
                - "ses:SendEmail"
                - "ses:VerifyEmailIdentity"
              Resource: "*"
            - Effect: "Allow"
              Action:
                - "sns:Publish"
                - "sns:OptInPhoneNumber"
                - "sns:CheckIfPhoneNumberIsOptedOut"
                - "sms-voice:DescribeOptedOutNumbers"
              Resource: "*"
            - Effect: "Deny"
              Action:
                - "sns:Publish"
              Resource: "arn:aws:sns:*:*:*"
NotificationLambdaFunction:
  Type: AWS::Lambda::Function
  Properties:
    Role: !GetAtt NotificationLambdaRole.Arn
    Runtime: python3.7
    Handler: index.lambda_handler
    Timeout: 300
    MemorySize: 3008
    Code:
      ZipFile: |
        import boto3
        import json
        import logging
        import datetime
        logger = logging.getLogger()
        logger.setLevel(logging.INFO)
        ses = boto3.client('ses')
```

```
          sns = boto3.client('sns')
          def check_value(target):
            if target:
              return True
            return False

          # Check whether email is verified. Only verified emails are allowed to send
emails to or from.
          def check_email(email):
            if not check_value(email):
              return False
            result = ses.get_identity_verification_attributes(Identities=[email])
            attr = result['VerificationAttributes']
            if (email not in attr or attr[email]['VerificationStatus'] != 'Success'):
                logging.info('Verification email for {} sent. You must have all the
emails verified before sending email.'.format(email))
                ses.verify_email_identity(EmailAddress=email)
                return False
            return True

          # Check whether the phone holder has opted out of receiving SMS messages from
your account
          def check_phone_number(phone_number):
            try:
              result = sns.check_if_phone_number_is_opted_out(phoneNumber=phone_number)
              if (result['isOptedOut']):
                  logger.info('phoneNumber {} is not opt in of receiving SMS messages.
Phone number must be opt in first.'.format(phone_number))
                  return False
              return True
            except Exception as e:
              logging.error('Your phone number {} must be in E.164 format in SSO.
Exception thrown: {}'.format(phone_number, e))
              return False

          def check_emails(emails):
            result = True
            for email in emails:
                if not check_email(email):
                    result = False
            return result

          def lambda_handler(event, context):
            logging.info('Received event: ' + json.dumps(event))
```

```
            nep = json.loads(event.get('notificationEventPayload'))
            alarm_state = nep['alarmState']
            default_msg = 'Alarm ' + alarm_state['stateName'] + '\n'
            timestamp =
 datetime.datetime.utcfromtimestamp(float(nep['stateUpdateTime'])/1000).strftime('%Y-
%m-%d %H:%M:%S')
            alarm_msg = "{} {} {} at {} UTC ".format(nep['alarmModelName'],
 nep.get('keyValue', 'Singleton'), alarm_state['stateName'], timestamp)
            default_msg += 'Sev: ' + str(nep['severity']) + '\n'
            if (alarm_state['ruleEvaluation']):
              property = alarm_state['ruleEvaluation']['simpleRule']['inputProperty']
              default_msg += 'Current Value: ' + str(property) + '\n'
              operator = alarm_state['ruleEvaluation']['simpleRule']['operator']
              threshold = alarm_state['ruleEvaluation']['simpleRule']['threshold']
              alarm_msg += '({} {} {})'.format(str(property), operator, str(threshold))
            default_msg += alarm_msg + '\n'

            emails = event.get('emailConfigurations', [])
            logger.info('Start Sending Emails')
            for email in emails:
              from_adr = email.get('from')
              to_adrs = email.get('to', [])
              cc_adrs = email.get('cc', [])
              bcc_adrs = email.get('bcc', [])
              msg = default_msg + '\n' + email.get('additionalMessage', '')
              subject = email.get('subject', alarm_msg)
              fa_ver = check_email(from_adr)
              tas_ver = check_emails(to_adrs)
              ccas_ver = check_emails(cc_adrs)
              bccas_ver = check_emails(bcc_adrs)
              if (fa_ver and tas_ver and ccas_ver and bccas_ver):
                ses.send_email(Source=from_adr,
                               Destination={'ToAddresses': to_adrs, 'CcAddresses':
 cc_adrs, 'BccAddresses': bcc_adrs},
                               Message={'Subject': {'Data': subject}, 'Body': {'Text':
 {'Data': msg}}})
                logger.info('Emails have been sent')

            logger.info('Start Sending SNS message to SMS')
            sns_configs = event.get('smsConfigurations', [])
            for sns_config in sns_configs:
              sns_msg = default_msg + '\n' + sns_config.get('additionalMessage', '')
              phone_numbers = sns_config.get('phoneNumbers', [])
              sender_id = sns_config.get('senderId')
```

```
            for phone_number in phone_numbers:
                if check_phone_number(phone_number):
                  if check_value(sender_id):
                    sns.publish(PhoneNumber=phone_number, Message=sns_msg,
MessageAttributes={'AWS.SNS.SMS.SenderID':{'DataType': 'String','StringValue':
sender_id}})
                  else:
                    sns.publish(PhoneNumber=phone_number, Message=sns_msg)
                  logger.info('SNS messages have been sent')
```

Verwenden der Lambda-Funktion von AWS IoT Events

Bei Alarmbenachrichtigungen können Sie die Lambda-Funktion verwenden, die von AWS IoT Events zur Verwaltung von Alarmbenachrichtigungen bereitgestellt wird.

Die folgenden Anforderungen gelten, wenn Sie die Lambda-Funktion von verwenden AWS IoT Events , um Ihre Alarmbenachrichtigungen zu verwalten:

- Sie müssen die E-Mail-Adresse, mit der die E-Mail-Benachrichtigungen gesendet werden, in Amazon Simple Email Service (Amazon SES) verifizieren. Weitere Informationen finden Sie unter Verifizieren der Identität einer E-Mail-Adresse im Amazon Simple Email Service Developer Guide.

  Wenn Sie einen Bestätigungslink erhalten, klicken Sie auf den Link, um Ihre E-Mail-Adresse zu verifizieren. Sie können auch in Ihrem Spam-Ordner nach einer Bestätigungs-E-Mail suchen.
- Wenn Ihr Alarm SMS-Benachrichtigungen sendet, müssen Sie die internationale E.164-Formatierung für Telefonnummern verwenden. Dieses Format enthält+`<country-calling-code><area-code><phone-number>`.

  Beispiele für Telefonnummern:

Land	Lokale Telefonnummer	E.164 formatierte Nummer
Vereinigte Staaten	206-555-0100	+12065550100
Großbritannien und Nordirland	020-1234-1234	+442012341234
Litauen	+601+12345	+37060112345

Um eine Landesvorwahl zu finden, gehen Sie zu countrycode.org.

Die von bereitgestellte Lambda-Funktion AWS IoT Events überprüft, ob Sie E.164-formatierte Telefonnummern verwenden. Die Telefonnummern werden jedoch nicht überprüft. Wenn Sie sicherstellen, dass Sie korrekte Telefonnummern eingegeben, aber keine SMS-Benachrichtigungen erhalten haben, können Sie sich an die Telefonanbieter wenden. Die Mobilfunkanbieter können die Nachrichten blockieren.

Verwalten Sie den Zugriff von Alarmempfängern auf das IAM Identity Center in AWS IoT Events

AWS IoT Events wird verwendet AWS IAM Identity Center , um den SSO-Zugriff von Alarmempfängern zu verwalten. Die Implementierung von IAM Identity Center für AWS IoT Events Benachrichtigungsempfänger kann die Sicherheit und das Benutzererlebnis verbessern. Damit der Alarm Benachrichtigungen an die Empfänger sendet, müssen Sie IAM Identity Center aktivieren und Empfänger zu Ihrem IAM Identity Center-Shop hinzufügen. Weitere Informationen finden Sie im AWS IAM Identity Center Benutzerhandbuch unter Benutzer hinzufügen.

Important

- Sie müssen dieselbe AWS Region für AWS IoT Events AWS Lambda, und IAM Identity Center auswählen.
- AWS Organizations unterstützen jeweils nur eine IAM Identity Center-Region. Wenn Sie IAM Identity Center in einer anderen Region verfügbar machen möchten, müssen Sie zuerst Ihre aktuelle IAM Identity Center-Konfiguration löschen. Weitere Informationen finden Sie unter IAM Identity Center-Regionaldaten im AWS IAM Identity Center Benutzerhandbuch.

Sicherheit in AWS IoT Events

Cloud-Sicherheit AWS hat höchste Priorität. Als AWS Kunde profitieren Sie von einer Rechenzentrums- und Netzwerkarchitektur, die darauf ausgelegt sind, die Anforderungen der sicherheitssensibelsten Unternehmen zu erfüllen.

Sicherheit ist eine gemeinsame Verantwortung von Ihnen AWS und Ihnen. Das Modell der geteilten Verantwortung beschreibt dies als Sicherheit der Cloud und Sicherheit in der Cloud:

- Sicherheit der Cloud — AWS ist verantwortlich für den Schutz der Infrastruktur, die AWS Dienste in der AWS Cloud ausführt. AWS bietet Ihnen auch Dienste, die Sie sicher nutzen können. Die Wirksamkeit unserer Sicherheitsfunktionen wird regelmäßig von externen Prüfern im Rahmen des AWS -Compliance-Programms getestet und überprüft. Weitere Informationen zu den Compliance-Programmen, die für gelten AWS IoT Events, finden Sie unter AWS Services nach Compliance-Programmen.
- Sicherheit in der Cloud — Ihre Verantwortung richtet sich nach dem AWS Dienst, den Sie nutzen. In Ihre Verantwortung fallen außerdem weitere Faktoren, wie z. B. die Vertraulichkeit der Daten, die Anforderungen Ihrer Organisation sowie geltende Gesetze und Vorschriften.

Diese Dokumentation hilft Ihnen zu verstehen, wie Sie das Modell der gemeinsamen Verantwortung bei der Nutzung anwenden können AWS IoT Events. In den folgenden Themen erfahren Sie, wie Sie die Konfiguration vornehmen AWS IoT Events , um Ihre Sicherheits- und Compliance-Ziele zu erreichen. Außerdem erfahren Sie, wie Sie andere AWS Dienste nutzen können, die Ihnen bei der Überwachung und Sicherung Ihrer AWS IoT Events Ressourcen helfen können.

Topics

Identitäts- und Zugriffsmanagement für AWS IoT Events

AWS Identity and Access Management (IAM) ist ein AWS Dienst, der einem Administrator hilft, den Zugriff auf AWS Ressourcen sicher zu kontrollieren. IAM-Administratoren kontrollieren, wer authentifiziert (angemeldet) und autorisiert werden kann (über Berechtigungen verfügt), um Ressourcen zu verwenden. AWS IoT Events IAM ist ein AWS Dienst, den Sie ohne zusätzliche Kosten nutzen können.

Themen

Zielgruppe

Wie Sie AWS Identity and Access Management (IAM) verwenden, hängt von Ihrer Rolle ab:

- Servicebenutzer – Fordern Sie von Ihrem Administrator Berechtigungen an, wenn Sie nicht auf Features zugreifen können (siehe Probleme mit AWS IoT Events Identität und Zugriff beheben).
- Serviceadministrator – Bestimmen Sie den Benutzerzugriff und stellen Sie Berechtigungsanfragen (siehe Wie AWS IoT Events funktioniert mit IAM).
- IAM-Administrator – Schreiben Sie Richtlinien zur Zugriffsverwaltung (siehe AWS IoT Events Beispiele für identitätsbasierte Richtlinien).

Authentifizierung mit Identitäten

Authentifizierung ist die Art und Weise, wie Sie sich AWS mit Ihren Identitätsdaten anmelden. Sie müssen sich als IAM-Benutzer authentifizieren oder eine IAM-Rolle annehmen. Root-Benutzer des AWS-Kontos

Sie können sich als föderierte Identität anmelden, indem Sie Anmeldeinformationen aus einer Identitätsquelle wie AWS IAM Identity Center (IAM Identity Center), Single Sign-On-Authentifizierung oder Anmeldeinformationen verwenden. Google/Facebook Weitere Informationen zum Anmelden finden Sie unter So melden Sie sich bei Ihrem AWS-Konto an im Benutzerhandbuch für AWS-Anmeldung .

AWS Bietet für den programmatischen Zugriff ein SDK und eine CLI zum kryptografischen Signieren von Anfragen. Weitere Informationen finden Sie unter AWS Signature Version 4 for API requests im IAM-Benutzerhandbuch.

AWS-Konto Root-Benutzer

Wenn Sie einen erstellen AWS-Konto, beginnen Sie mit einer Anmeldeidentität, dem sogenannten AWS-Konto Root-Benutzer, der vollständigen Zugriff auf alle AWS-Services Ressourcen hat. Wir raten ausdrücklich davon ab, den Root-Benutzer für Alltagsaufgaben zu verwenden. Eine Liste der Aufgaben, für die Sie sich als Root-Benutzer anmelden müssen, finden Sie unter Tasks that require root user credentials im IAM-Benutzerhandbuch.

IAM-Benutzer und -Gruppen

Ein IAM-Benutzer ist eine Identität mit bestimmten Berechtigungen für eine einzelne Person oder Anwendung. Wir empfehlen die Verwendung temporärer Anmeldeinformationen anstelle von IAM-Benutzern mit langfristigen Anmeldeinformationen. Weitere Informationen finden Sie im IAM-Benutzerhandbuch unter Erfordern, dass menschliche Benutzer für den Zugriff AWS mithilfe temporärer Anmeldeinformationen einen Verbund mit einem Identitätsanbieter verwenden müssen.

Eine IAM-Gruppe spezifiziert eine Sammlung von IAM-Benutzern und erleichtert die Verwaltung von Berechtigungen für große Gruppen von Benutzern. Weitere Informationen finden Sie unter Anwendungsfälle für IAM-Benutzer im IAM-Benutzerhandbuch.

IAM-Rollen

Eine IAM-Rolle ist eine Identität mit spezifischen Berechtigungen, die temporäre Anmeldeinformationen bereitstellt. Sie können eine Rolle übernehmen, indem Sie von einer Benutzer- zu einer IAM-Rolle (Konsole) wechseln oder indem Sie eine AWS Oder-API-Operation AWS CLI aufrufen. Weitere Informationen finden Sie unter Methoden, um eine Rolle zu übernehmen im IAM-Benutzerhandbuch.

IAM-Rollen sind nützlich für den Verbundbenutzer-Zugriff, temporäre IAM-Benutzerberechtigungen, kontoübergreifenden Zugriff, serviceübergreifenden Zugriff und Anwendungen, die auf Amazon EC2

laufen. Weitere Informationen finden Sie unter Kontoübergreifender Ressourcenzugriff in IAM im IAM-Benutzerhandbuch.

Verwalten des Zugriffs mit Richtlinien

Sie kontrollieren den Zugriff, AWS indem Sie Richtlinien erstellen und diese an AWS Identitäten oder Ressourcen anhängen. Eine Richtlinie definiert Berechtigungen, wenn sie mit einer Identität oder Ressource verknüpft sind. AWS bewertet diese Richtlinien, wenn ein Principal eine Anfrage stellt. Die meisten Richtlinien werden AWS als JSON-Dokumente gespeichert. Weitere Informationen zu JSON-Richtliniendokumenten finden Sie unter Übersicht über JSON-Richtlinien im IAM-Benutzerhandbuch.

Mit Hilfe von Richtlinien legen Administratoren fest, wer Zugriff auf was hat, indem sie definieren, welches Prinzipal welche Aktionen auf welchen Ressourcenund unter welchen Bedingungendurchführen darf.

Standardmäßig haben Benutzer, Gruppen und Rollen keine Berechtigungen. Ein IAM-Administrator erstellt IAM-Richtlinien und fügt sie zu Rollen hinzu, die die Benutzer dann übernehmen können. IAM-Richtlinien definieren Berechtigungen unabhängig von der Methode, die zur Ausführung der Operation verwendet wird.

Identitätsbasierte Richtlinien

Identitätsbasierte Richtlinien sind JSON-Berechtigungsrichtliniendokumente, die Sie einer Identität (Benutzer, Gruppe oder Rolle) anfügen können. Diese Richtlinien steuern, welche Aktionen Identitäten für welche Ressourcen und unter welchen Bedingungen ausführen können. Informationen zum Erstellen identitätsbasierter Richtlinien finden Sie unter Definieren benutzerdefinierter IAM-Berechtigungen mit vom Kunden verwalteten Richtlinien im IAM-Benutzerhandbuch.

Identitätsbasierte Richtlinien können Inline-Richtlinien (direkt in eine einzelne Identität eingebettet) oder verwaltete Richtlinien (eigenständige Richtlinien, die mit mehreren Identitäten verbunden sind) sein. Informationen dazu, wie Sie zwischen verwalteten und Inline-Richtlinien wählen, finden Sie unter Choose between managed policies and inline policies im IAM-Benutzerhandbuch.

Weitere Richtlinientypen

AWS unterstützt zusätzliche Richtlinientypen, mit denen die maximalen Berechtigungen festgelegt werden können, die durch gängigere Richtlinientypen gewährt werden:

- Berechtigungsgrenzen – Eine Berechtigungsgrenze legt die maximalen Berechtigungen fest, die eine identitätsbasierte Richtlinie einer IAM-Entität erteilen kann. Weitere Informationen finden Sie unter Berechtigungsgrenzen für IAM-Entitäten im -IAM-Benutzerhandbuch.
- Richtlinien zur Dienstkontrolle (SCPs) — Geben Sie die maximalen Berechtigungen für eine Organisation oder Organisationseinheit in an AWS Organizations. Weitere Informationen finden Sie unter Service-Kontrollrichtlinien im AWS Organizations -Benutzerhandbuch.
- Richtlinien zur Ressourcenkontrolle (RCPs) — Legen Sie die maximal verfügbaren Berechtigungen für Ressourcen in Ihren Konten fest. Weitere Informationen finden Sie im AWS Organizations Benutzerhandbuch unter Richtlinien zur Ressourcenkontrolle (RCPs).
- Sitzungsrichtlinien – Sitzungsrichtlinien sind erweiterte Richtlinien, die als Parameter übergeben werden, wenn Sie eine temporäre Sitzung für eine Rolle oder einen Verbundbenutzer erstellen. Weitere Informationen finden Sie unter Sitzungsrichtlinien im IAM-Benutzerhandbuch.

Mehrere Richtlinientypen

Wenn für eine Anfrage mehrere Arten von Richtlinien gelten, sind die daraus resultierenden Berechtigungen schwieriger zu verstehen. Informationen darüber, wie AWS bestimmt wird, ob eine Anfrage zulässig ist, wenn mehrere Richtlinientypen betroffen sind, finden Sie unter Bewertungslogik für Richtlinien im IAM-Benutzerhandbuch.

Weitere Informationen zu Identitäts- und Zugriffsmanagement

Weitere Informationen zur Identitäts- und Zugriffsverwaltung für AWS IoT Events finden Sie auf den folgenden Seiten:

- Wie AWS IoT Events funktioniert mit IAM
- Probleme mit AWS IoT Events Identität und Zugriff beheben

Wie AWS IoT Events funktioniert mit IAM

Bevor Sie IAM verwenden, um den Zugriff auf zu verwalten AWS IoT Events, sollten Sie wissen, mit welchen IAM-Funktionen Sie verwenden können. AWS IoT EventsEinen allgemeinen Überblick darüber, wie AWS IoT Events und andere AWS Dienste mit IAM funktionieren, finden Sie im AWS IAM-Benutzerhandbuch unter Dienste, die mit IAM funktionieren.

Themen

- AWS IoT Events identitätsbasierte Richtlinien
- AWS IoT Events ressourcenbasierte Richtlinien
- Autorisierung auf der Grundlage von Tags AWS IoT Events
- AWS IoT Events IAM-Rollen

AWS IoT Events identitätsbasierte Richtlinien

Mit identitätsbasierten IAM-Richtlinien können Sie angeben, welche Aktionen und Ressourcen erteilt oder abgelehnt werden. Darüber hinaus können Sie die Bedingungen festlegen, unter denen Aktionen zugelassen oder abgelehnt werden. AWS IoT Events unterstützt bestimmte Aktionen, Ressourcen und Bedingungsschlüssel. Informationen zu sämtlichen Elementen, die Sie in einer JSON-Richtlinie verwenden, finden Sie in der IAM-Referenz für JSON-Richtlinienelemente im IAM-Benutzerhandbuch.

Aktionen

Das Element `Action` einer identitätsbasierten IAM-Richtlinie beschreibt die spezifischen Aktionen, die von der Richtlinie zugelassen oder abgelehnt werden. Richtlinienaktionen haben normalerweise denselben Namen wie der zugehörige AWS API-Vorgang. Die Aktion wird in einer Richtlinie verwendet, um Berechtigungen zur Durchführung der zugehörigen Aktion zu gewähren.

Bei Richtlinienaktionen wird vor der Aktion das folgende Präfix AWS IoT Events verwendet:`iotevents:`. Um beispielsweise jemandem die Erlaubnis zu erteilen, mit der AWS IoT Events `CreateInput` API-Operation eine AWS IoT Events Eingabe zu erstellen, nehmen Sie die `iotevents:CreateInput` Aktion in seine Richtlinie auf. Um jemandem die Erlaubnis zu erteilen, im Rahmen des AWS IoT Events `BatchPutMessage` API-Vorgangs eine Eingabe zu senden, nehmen Sie die `iotevents-data:BatchPutMessage` Aktion in seine Richtlinie auf. Richtlinienerklärungen müssen `Action` entweder ein `NotAction` Oder-Element enthalten. AWS IoT Events definiert eigene Aktionen, die Aufgaben beschreiben, die Sie mit diesem Dienst ausführen können.

Um mehrere Aktionen in einer einzigen Anweisung anzugeben, trennen Sie sie wie folgt durch Kommata:

```
"Action": [
      "iotevents:action1",
      "iotevents:action2"
```

Sie können auch Platzhalter verwenden, um mehrere Aktionen anzugeben. Beispielsweise können Sie alle Aktionen festlegen, die mit dem Wort `Describe` beginnen, einschließlich der folgenden Aktion:

```
"Action": "iotevents:Describe*"
```

Eine Liste der AWS IoT Events [Aktionen finden Sie AWS IoT Events im IAM-Benutzerhandbuch unter Definierte Aktionen von](#).

Ressourcen

Das Element `Resource` gibt die Objekte an, auf die die Aktion angewendet wird. Anweisungen müssen entweder ein `Resource`- oder ein `NotResource`-Element enthalten. Sie geben eine Ressource unter Verwendung eines ARN oder eines Platzhalters (*) an, um anzugeben, dass die Anweisung für alle Ressourcen gilt.

Die AWS IoT Events Detektormodellressource hat den folgenden ARN:

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

Weitere Informationen zum Format von ARNs finden Sie unter [Identifizieren von AWS Ressourcen mit Amazon-Ressourcennamen (ARNs)](#).

Um beispielsweise das `Foobar` Detektormodell in Ihrer Anweisung anzugeben, verwenden Sie den folgenden ARN:

```
"Resource": "arn:aws:iotevents:us-east-1:123456789012:detectorModel/Foobar"
```

Um alle Instances anzugeben, die zu einem bestimmten Konto gehören, verwenden Sie den Platzhalter (*):

```
"Resource": "arn:aws:iotevents:us-east-1:123456789012:detectorModel/*"
```

Einige AWS IoT Events Aktionen, z. B. die zum Erstellen von Ressourcen, können nicht für eine bestimmte Ressource ausgeführt werden. In diesen Fällen müssen Sie den Platzhalter (*) verwenden.

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

Einige AWS IoT Events API-Aktionen umfassen mehrere Ressourcen. Verweist beispielsweise in seinen Bedingungsanweisungen `CreateDetectorModel` auf Eingaben, sodass ein Benutzer über Berechtigungen zur Verwendung der Eingabe und des Detektormodells verfügen muss. Um mehrere Ressourcen in einer einzigen Anweisung anzugeben, trennen Sie sie ARNs durch Kommas.

```
"Resource": [
      "resource1",
      "resource2"
```

Eine Liste der AWS IoT Events Ressourcentypen und ihrer ARNs Eigenschaften finden Sie AWS IoT Events im IAM-Benutzerhandbuch unter Defined by (Ressourcen definiert von). Informationen zu den Aktionen, mit denen Sie den ARN einzelner Ressourcen angeben können, finden Sie unter Von AWS IoT Events definierte Aktionen.

Bedingungsschlüssel

Mithilfe des Elements `Condition`(oder des Blocks `Condition`) können Sie die Bedingungen angeben, unter denen eine Anweisung wirksam ist. Das Element `Condition` ist optional. Sie können bedingte Ausdrücke erstellen, die Bedingungs-Operatoren verwenden, z. B. ist gleich oder kleiner als, damit die Bedingung in der Richtlinie mit Werten in der Anforderung übereinstimmt.

Wenn Sie mehrere `Condition`-Elemente in einer Anweisung oder mehrere Schlüssel in einem einzelnen `Condition`-Element angeben, wertet AWS diese mittels einer logischen `AND`-Operation aus. Wenn Sie mehrere Werte für einen einzelnen Bedingungsschlüssel angeben, AWS wertet die Bedingung mithilfe einer logischen `OR` Operation aus. Alle Bedingungen müssen erfüllt sein, bevor die Berechtigungen für die Anweisung erteilt werden.

Sie können bei der Angabe von Bedingungen auch Platzhaltervariablen verwenden. Beispielsweise können Sie einem Benutzer die Berechtigung für den Zugriff auf eine Ressource nur dann gewähren, wenn sie mit dessen Benutzernamen gekennzeichnet ist. Weitere Informationen finden Sie unter IAM-Richtlinienelemente: Variablen und Tags (Markierungen) im IAM-Benutzerhandbuch.

AWS IoT Events stellt keine dienstspezifischen Bedingungsschlüssel bereit, unterstützt aber die Verwendung einiger globaler Bedingungsschlüssel. Eine Übersicht aller AWS globalen Bedingungsschlüssel finden Sie unter Kontextschlüssel für AWS globale Bedingungen im IAM-Benutzerhandbuch.“

Beispiele

Beispiele für AWS IoT Events identitätsbasierte Richtlinien finden Sie unter. AWS IoT Events Beispiele für identitätsbasierte Richtlinien

AWS IoT Events ressourcenbasierte Richtlinien

AWS IoT Events unterstützt keine ressourcenbasierten Richtlinien.“ Ein Beispiel für eine detaillierte Seite zu ressourcenbasierten Richtlinien finden Sie unter https://docs.aws.amazon.com/lambda/latest/dg/access-control-resource-based.html.

Autorisierung auf der Grundlage von Tags AWS IoT Events

Sie können Tags an AWS IoT Events Ressourcen anhängen oder Tags in einer Anfrage an übergeben AWS IoT Events. Um den Zugriff auf der Grundlage von Tags zu steuern, geben Sie im Bedingungselement einer Richtlinie Tag-Informationen an, indem Sie die Schlüssel `iotevents:ResourceTag/`*`key-name`*, `aws:RequestTag/`*`key-name`*, oder Bedingung `aws:TagKeys` verwenden. Weitere Informationen über das Markieren von AWS IoT Events - Ressourcen mit Tags finden Sie unter Verschlagworten Sie Ihre Ressourcen AWS IoT Events.

Ein Beispiel für eine identitätsbasierte Richtlinie zur Einschränkung des Zugriffs auf eine Ressource auf der Grundlage der Markierungen dieser Ressource finden Sie unter Auf Tags basierende AWS IoT Events Eingaben anzeigen.

AWS IoT Events IAM-Rollen

Eine IAM-Rolle ist eine Entität innerhalb von Ihnen AWS-Konto , die über bestimmte Berechtigungen verfügt.

Verwenden temporärer Anmeldeinformationen mit AWS IoT Events

Sie können temporäre Anmeldeinformationen verwenden, um sich über einen Verbund anzumelden, eine IAM-Rolle anzunehmen oder eine kontenübergreifende Rolle anzunehmen. Sie erhalten temporäre Sicherheitsanmeldedaten, indem Sie AWS -Security-Token-Service (AWS STS) API-Operationen wie AssumeRoleoder aufrufen GetFederationToken.

AWS IoT Events unterstützt die Verwendung temporärer Anmeldeinformationen nicht.

Service-verknüpfte Rollen

Mit dienstbezogenen Rollen können AWS Dienste auf Ressourcen in anderen Diensten zugreifen, um eine Aktion in Ihrem Namen auszuführen. Serviceverknüpfte Rollen werden in Ihrem IAM-Konto angezeigt und gehören zum Service. Ein IAM-Administrator kann die Berechtigungen für serviceverknüpfte Rollen anzeigen, aber nicht bearbeiten.

AWS IoT Events unterstützt keine dienstbezogenen Rollen.

Servicerollen

Dieses Feature ermöglicht einem Service das Annehmen einer Servicerolle in Ihrem Namen. Diese Rolle gewährt dem Service Zugriff auf Ressourcen in anderen Diensten, um eine Aktion in Ihrem Namen auszuführen. Servicerollen werden in Ihrem IAM-Konto angezeigt und gehören zum Konto. Dies bedeutet, dass ein IAM-Administrator die Berechtigungen für diese Rolle ändern kann. Dies kann jedoch die Funktionalität des Dienstes beeinträchtigen.

AWS IoT Events unterstützt Servicerollen.

AWS IoT Events Beispiele für identitätsbasierte Richtlinien

Standardmäßig sind Benutzer und Rollen nicht berechtigt, AWS IoT Events Ressourcen zu erstellen oder zu ändern. Sie können auch keine Aufgaben mit der AWS-Managementkonsole AWS CLI, oder AWS API ausführen. Ein IAM-Administrator muss IAM-Richtlinien erstellen, die Benutzern und Rollen die Berechtigung zum Ausführen bestimmter API-Operationen für die angegebenen Ressourcen gewähren, die diese benötigen. Der Administrator muss diese Richtlinien anschließend den -Benutzern oder -Gruppen anfügen, die diese Berechtigungen benötigen.

Informationen dazu, wie Sie unter Verwendung dieser beispielhaften JSON-Richtliniendokumente eine identitätsbasierte IAM-Richtlinie erstellen, finden Sie unter Erstellen von Richtlinien auf der JSON-Registerkarte im IAM-Benutzerhandbuch.

Themen

- Best Practices für Richtlinien
- Verwenden der Konsole AWS IoT Events
- Erlauben Sie Benutzern, ihre eigenen Berechtigungen in einzusehen AWS IoT Events
- Greifen Sie auf eine Eingabe zu AWS IoT Events
- Auf Tags basierende AWS IoT Events Eingaben anzeigen

Best Practices für Richtlinien

Identitätspolitiken sind sehr wirksam. Sie bestimmen, ob jemand AWS IoT Events Ressourcen in Ihrem Konto erstellen, darauf zugreifen oder sie löschen kann. Dies kann zusätzliche Kosten für Ihr verursachen AWS-Konto. Beachten Sie beim Erstellen oder Bearbeiten identitätsbasierter Richtlinien die folgenden Richtlinien und Empfehlungen:

- Erste Schritte mit AWS verwalteten Richtlinien — Damit Sie AWS IoT Events schnell mit der Nutzung von AWS verwalteten Richtlinien beginnen können, geben Sie Ihren Mitarbeitern die erforderlichen Berechtigungen. Diese Richtlinien sind bereits in Ihrem Konto verfügbar und werden von AWS. Weitere Informationen finden Sie im IAM-Benutzerhandbuch unter Erste Schritte zur Nutzung von Berechtigungen mit AWS verwalteten Richtlinien.
- Gewähren Sie die geringstmöglichen Berechtigungen – Gewähren Sie beim Erstellen benutzerdefinierter Richtlinien nur die Berechtigungen, die zum Ausführen einer Aufgabe erforderlich sind. Beginnen Sie mit einem Mindestsatz von Berechtigungen und gewähren Sie zusätzliche Berechtigungen wie erforderlich. Dies ist sicherer, als mit Berechtigungen zu beginnen, die zu weit gefasst sind, und dann später zu versuchen, sie zu begrenzen. Weitere Informationen finden Sie unter Gewähren von geringsten Rechten im IAM-Benutzerhandbuch.
- MFA für sensible Operationen aktivieren — Für zusätzliche Sicherheit müssen Benutzer die Multi-Faktor-Authentifizierung (MFA) verwenden, um auf vertrauliche Ressourcen oder API-Operationen zuzugreifen. Weitere Informationen finden Sie unter Verwenden der Multi-Faktor-Authentifizierung (MFA) in AWS im IAM-Benutzerhandbuch.
- Verwenden Sie Richtlinienbedingungen, um zusätzliche Sicherheit zu bieten – Definieren Sie die Bedingungen, unter denen Ihre identitätsbasierten Richtlinien den Zugriff auf eine Ressource zulassen, soweit praktikabel. Beispielsweise können Sie Bedingungen schreiben, die eine Reihe von zulässigen IP-Adressen festlegen, von denen eine Anforderung stammen muss. Sie können auch Bedingungen schreiben, die Anforderungen nur innerhalb eines bestimmten Datums- oder Zeitbereichs zulassen oder die Verwendung von SSL oder MFA fordern. Weitere Informationen finden Sie unter IAM-JSON-Richtlinienelemente: Bedingung im IAM-Benutzerhandbuch.

Verwenden der Konsole AWS IoT Events

Um auf die AWS IoT Events Konsole zugreifen zu können, benötigen Sie ein Mindestmaß an Berechtigungen. Diese Berechtigungen müssen es Ihnen ermöglichen, Details zu den AWS IoT Events Ressourcen in Ihrem aufzulisten und anzuzeigen AWS-Konto. Wenn Sie eine identitätsbasierte Richtlinie erstellen, die strenger ist als die mindestens erforderlichen Berechtigungen, funktioniert die Konsole nicht wie vorgesehen für Entitäten (Benutzer oder Rollen) mit dieser Richtlinie.

Um sicherzustellen, dass diese Entitäten die AWS IoT Events Konsole weiterhin verwenden können, fügen Sie den Entitäten außerdem die folgende AWS verwaltete Richtlinie hinzu. Weitere Informationen finden Sie unter Hinzufügen von Berechtigungen für einen Benutzer im IAM-Benutzerhandbuch:

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iotevents:BatchPutMessage",
                "iotevents:BatchUpdateDetector",
                "iotevents:CreateDetectorModel",
                "iotevents:CreateInput",
                "iotevents:DeleteDetectorModel",
                "iotevents:DeleteInput",
                "iotevents:DescribeDetector",
                "iotevents:DescribeDetectorModel",
                "iotevents:DescribeInput",
                "iotevents:DescribeLoggingOptions",
                "iotevents:ListDetectorModelVersions",
                "iotevents:ListDetectorModels",
                "iotevents:ListDetectors",
                "iotevents:ListInputs",
                "iotevents:ListTagsForResource",
                "iotevents:PutLoggingOptions",
                "iotevents:TagResource",
                "iotevents:UntagResource",
                "iotevents:UpdateDetectorModel",
                "iotevents:UpdateInput",
                "iotevents:UpdateInputRouting"
            ],
            "Resource": "arn:aws:iotevents:us-
east-1:123456789012:detectorModel/your-detector-model-name",
    "Resource": "arn:aws:iotevents:us-east-1:123456789012:input/your-input-name"
        }
    ]
}
```

Sie müssen Benutzern, die nur die API AWS CLI oder die AWS API aufrufen, keine Mindestberechtigungen für die Konsole gewähren. Stattdessen sollten Sie nur Zugriff auf die Aktionen zulassen, die der API-Operation entsprechen, die Sie ausführen möchten.

Erlauben Sie Benutzern, ihre eigenen Berechtigungen in einzusehen AWS IoT Events

In diesem Beispiel wird gezeigt, wie Sie eine Richtlinie erstellen, die -Benutzern die Berechtigung zum Anzeigen der Inline-Richtlinien und verwalteten Richtlinien gewährt, die ihrer Benutzeridentität angefügt sind. Benutzern die Möglichkeit zu geben, ihre eigenen IAM-Berechtigungen einzusehen, ist nützlich, um das Sicherheitsbewusstsein zu erhöhen und Self-Service-Funktionen zu nutzen. Diese Richtlinie umfasst Berechtigungen zum Ausführen dieser Aktion auf der Konsole oder programmgesteuert mithilfe der OR-API. AWS CLI AWS

JSON

```
{
       "Version":"2012-10-17",
       "Statement": [
           {
               "Sid": "ViewOwnUserInfo",
               "Effect": "Allow",
               "Action": [
                   "iam:GetUserPolicy",
                   "iam:ListGroupsForUser",
                   "iam:ListAttachedUserPolicies",
                   "iam:ListUserPolicies",
                   "iam:GetUser"
               ],
               "Resource": [
                   "arn:aws:iam::*:user/${aws:username}"
               ]
           },
           {
               "Sid": "NavigateInConsole",
               "Effect": "Allow",
               "Action": [
                   "iam:GetGroupPolicy",
                   "iam:GetPolicyVersion",
                   "iam:GetPolicy",
                   "iam:ListAttachedGroupPolicies",
                   "iam:ListGroupPolicies",
                   "iam:ListPolicyVersions",
                   "iam:ListPolicies",
                   "iam:ListUsers"
               ],
               "Resource": "*"
```

```
            }
        ]
    }
```

Greifen Sie auf eine Eingabe zu AWS IoT Events

Eine detaillierte Zugriffskontrolle auf AWS IoT Events Eingaben ist wichtig, um die Sicherheit in Umgebungen mit mehreren Benutzern oder mehreren Teams aufrechtzuerhalten. In diesem Abschnitt wird gezeigt, wie Sie IAM-Richtlinien erstellen, die den Zugriff auf bestimmte AWS IoT Events Eingaben gewähren und gleichzeitig den Zugriff auf andere einschränken.

In diesem Beispiel können Sie einem Benutzer AWS-Konto Zugriff auf eine Ihrer AWS IoT Events Eingaben gewähren. `exampleInput` Sie können dem Benutzer auch erlauben, Eingaben hinzuzufügen, zu aktualisieren und zu löschen.

Die Richtlinie gewährt dem Benutzer die Berechtigungen `iotevents:ListInputs`, `iotevents:DescribeInput`, `iotevents:CreateInput`, `iotevents:DeleteInput` und `iotevents:UpdateInput`. Ein Beispiel für den Amazon Simple Storage Service (Amazon S3), der Benutzern Berechtigungen erteilt und sie mithilfe der Konsole testet, finden Sie unter Steuern des Zugriffs auf einen Bucket mit Benutzerrichtlinien.

JSON

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Sid":"ListInputsInConsole",
         "Effect":"Allow",
         "Action":[
            "iotevents:ListInputs"
         ],
         "Resource":"arn:aws:iotevents:us-east-2:123456789012:input/*"
      },
      {
         "Sid":"ViewSpecificInputInfo",
         "Effect":"Allow",
         "Action":[
            "iotevents:DescribeInput"
         ],
```

```
            "Resource":"arn:aws:iotevents:us-east-1:123456789012:input/inputName"
        },
        {
            "Sid":"ManageInputs",
            "Effect":"Allow",
            "Action":[
                "iotevents:CreateInput",
                "iotevents:DeleteInput",
                "iotevents:DescribeInput",
                "iotevents:ListInputs",
                "iotevents:UpdateInput"
            ],
            "Resource":"arn:aws:iotevents:us-east-1:123456789012:input/*"
        }
    ]
}
```

Auf Tags basierende AWS IoT Events Eingaben anzeigen

Mithilfe von Tags können Sie AWS IoT Events Ressourcen organisieren. Sie können Bedingungen in Ihrer identitätsbasierten Richtlinie verwenden, um den Zugriff auf AWS IoT Events Ressourcen anhand von Stichwörtern zu steuern. Dieses Beispiel zeigt, wie Sie eine Richtlinie erstellen können, die das Anzeigen von ermöglicht. *input* Die Berechtigung wird jedoch nur gewährt, wenn der Wert des *input*-Tags `Owner` der Name des Benutzers ist. Diese Richtlinie gewährt auch die Berechtigungen, die für die Ausführung dieser Aktion auf der Konsole erforderlich sind.

JSON

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

Sie können diese Richtlinie den -Benutzern in Ihrem Konto zuweisen. Wenn ein Benutzer mit dem Namen `richard-roe` versucht AWS IoT Events *`input`*, eine aufzurufen, *`input`* muss er mit `0wner=richard-roe` oder markiert werden`owner=richard-roe`. Andernfalls wird der Zugriff abgelehnt. Der Tag-Schlüssel `0wner` der Bedingung stimmt sowohl mit `0wner` als auch mit `owner` überein, da die Namen von Bedingungsschlüsseln nicht zwischen Groß- und Kleinschreibung unterscheiden. Weitere Informationen finden Sie unter IAM-JSON-Richtlinienelemente: Bedingung im IAM-Benutzerhandbuch.

Dienstübergreifende verwirrte Stellvertreterprävention für AWS IoT Events

> **Note**
>
> - Mit dem AWS IoT Events Dienst können Sie Rollen nur verwenden, um Aktionen in demselben Konto zu starten, in dem eine Ressource erstellt wurde. Dies hilft, einen verwirrten stellvertretenden Angriff in zu verhindern AWS IoT Events.
> - Diese Seite dient als Referenz, um zu erfahren, wie das Problem mit dem verwirrten Stellvertreter funktioniert und verhindert werden kann, falls kontoübergreifende Ressourcen für den AWS IoT Events Dienst zugelassen wurden.

Das Problem des verwirrten Stellvertreters ist ein Sicherheitsproblem, bei dem eine Entität, die keine Berechtigung zur Durchführung einer Aktion hat, eine privilegiertere Entität zur Durchführung der Aktion zwingen kann. In: AWS Dienststellenübergreifender Identitätswechsel kann zu einem Problem mit dem verwirrten Stellvertreter führen.

Ein dienstübergreifender Identitätswechsel kann auftreten, wenn ein Dienst (der Anruf-Dienst) einen anderen Dienst anruft (den aufgerufenen Dienst). Der Anruf-Dienst kann so manipuliert werden, dass er seine Berechtigungen verwendet, um auf die Ressourcen eines anderen Kunden zu reagieren, auf die er sonst nicht zugreifen dürfte. Um dies zu verhindern, AWS bietet Tools, mit denen Sie Ihre

Daten für alle Dienste mit Dienstprinzipalen schützen können, denen Zugriff auf Ressourcen in Ihrem Konto gewährt wurde.

Wir empfehlen, die Kontextschlüssel `aws:SourceArn`und die `aws:SourceAccount`globalen Bedingungsschlüssel in Ressourcenrichtlinien zu verwenden, um die Berechtigungen einzuschränken, AWS IoT Events die der Ressource einen anderen Dienst gewähren. Wenn der `aws:SourceArn`-Wert die Konto-ID nicht enthält, z. B. einen Amazon-S3-Bucket-ARN, müssen Sie beide globale Bedingungskontextschlüssel verwenden, um Berechtigungen einzuschränken. Wenn Sie beide globale Bedingungskontextschlüssel verwenden und der `aws:SourceArn`-Wert die Konto-ID enthält, müssen der `aws:SourceAccount`-Wert und das Konto im `aws:SourceArn`-Wert dieselbe Konto-ID verwenden, wenn sie in der gleichen Richtlinienanweisung verwendet wird.

Verwenden Sie `aws:SourceArn`, wenn Sie nur eine Ressource mit dem betriebsübergreifenden Zugriff verknüpfen möchten. Verwenden Sie `aws:SourceAccount`, wenn Sie zulassen möchten, dass Ressourcen in diesem Konto mit der betriebsübergreifenden Verwendung verknüpft werden. Der Wert von `aws:SourceArn` muss dem Detektor- oder Alarmmodell entsprechen, das der `sts:AssumeRole` Anfrage zugeordnet ist.

Der effektivste Weg, um sich vor dem Confused-Deputy-Problem zu schützen, ist die Verwendung des globalen Bedingungskontext-Schlüssels `aws:SourceArn` mit dem vollständigen ARN der Ressource. Wenn Sie den vollständigen ARN der Ressource nicht kennen oder wenn Sie mehrere Ressourcen angeben, verwenden Sie den globalen Bedingungskontext-Schlüssel `aws:SourceArn` mit Platzhaltern (*) für die unbekannten Teile des ARN. Beispiel, `arn:aws:`*`iotevents`*`:*:`*`123456789012`*`:*`.

Die folgenden Beispiele zeigen, wie Sie die Kontextschlüssel `aws:SourceArn` und die `aws:SourceAccount` globale Bedingung verwenden können, AWS IoT Events um das Problem des verwirrten Stellvertreters zu vermeiden.

Themen

- Beispiel: Sicherer Zugriff auf ein AWS IoT Events Detektormodell
- Beispiel: Sicherer Zugriff auf ein AWS IoT Events Alarmmodell
- Beispiel: Greifen Sie auf eine AWS IoT Events Ressource in einer bestimmten Region zu
- Beispiel: Konfigurieren Sie die Protokollierungsoptionen für AWS IoT Events

Beispiel: Sicherer Zugriff auf ein AWS IoT Events Detektormodell

Dieses Beispiel zeigt, wie Sie eine IAM-Richtlinie erstellen, die den Zugriff auf ein bestimmtes Detektormodell in sicher gewährt. AWS IoT Events Die Richtlinie verwendet Bedingungen, um sicherzustellen, dass nur das angegebene AWS Konto und der angegebene AWS IoT Events Dienst die Rolle übernehmen können, wodurch eine zusätzliche Sicherheitsebene hinzugefügt wird. In diesem Beispiel kann die Rolle nur auf das angegebene Detektormodell zugreifen*WindTurbine01*.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": [
                    "iotevents.amazonaws.com"
                ]
            },
            "Action": "sts:AssumeRole",
            "Condition": {
                "StringEquals": {
                "aws:SourceAccount": "123456789012"
                },
                "ArnEquals": {
                "aws:SourceArn": "arn:aws:iotevents:us-
east-1:123456789012:detectorModel/WindTurbine01"
                }
            }
        }
    ]
}
```

Beispiel: Sicherer Zugriff auf ein AWS IoT Events Alarmmodell

Dieses Beispiel zeigt, wie eine IAM-Richtlinie erstellt wird, die den sicheren AWS IoT Events Zugriff auf Alarmmodelle ermöglicht. Die Richtlinie verwendet Bedingungen, um sicherzustellen, dass nur das angegebene AWS Konto und der angegebene AWS IoT Events Dienst die Rolle übernehmen können.

In diesem Beispiel kann die Rolle auf jedes Alarmmodell innerhalb des angegebenen AWS Kontos zugreifen, wie durch den * Platzhalter im Alarmmodell-ARN angegeben. Die `aws:SourceArn` Bedingungen `aws:SourceAccount` und die Bedingungen wirken zusammen, um das Problem des verwirrten Stellvertreters zu verhindern.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": [
                    "iotevents.amazonaws.com"
                ]
            },
            "Action": "sts:AssumeRole",
            "Condition": {
                "StringEquals": {
                "aws:SourceAccount": "123456789012"
                },
                "ArnEquals": {
                "aws:SourceArn": "arn:aws:iotevents:us-
east-1:123456789012:alarmModel/*"
                }
            }
        }
    ]
}
```

Beispiel: Greifen Sie auf eine AWS IoT Events Ressource in einer bestimmten Region zu

Dieses Beispiel zeigt, wie eine IAM-Rolle für den Zugriff auf AWS IoT Events Ressourcen in einer bestimmten AWS Region konfiguriert wird. Indem Sie ARNs in Ihren IAM-Richtlinien regionsspezifische Angaben verwenden, können Sie den Zugriff auf AWS IoT Events Ressourcen in verschiedenen geografischen Gebieten einschränken. Dieser Ansatz kann dazu beitragen, Sicherheit

und Compliance in Bereitstellungen in mehreren Regionen aufrechtzuerhalten. Die Region in diesem Beispiel ist. *us-east-1*

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": [
                    "iotevents.amazonaws.com"
                ]
            },
            "Action": "sts:AssumeRole",
            "Condition": {
                "StringEquals": {
                "aws:SourceAccount": "123456789012"
                },
                "ArnEquals": {
                "aws:SourceArn": "arn:aws:iotevents:us-east-1:123456789012:*"
                }
            }
        }
    ]
}
```

Beispiel: Konfigurieren Sie die Protokollierungsoptionen für AWS IoT Events

Die korrekte Protokollierung ist wichtig für die Überwachung, das Debuggen und die Prüfung Ihrer AWS IoT Events Anwendungen. Dieser Abschnitt bietet einen Überblick über die Protokollierungsoptionen, die in AWS IoT Events verfügbar sind.

Dieses Beispiel zeigt, wie eine IAM-Rolle konfiguriert wird, die es ermöglicht, Daten AWS IoT Events in CloudWatch Logs zu protokollieren. Die Verwendung von Platzhaltern (*) im Ressourcen-ARN ermöglicht eine umfassende Protokollierung in Ihrer gesamten AWS IoT Events Infrastruktur.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": [
                    "iotevents.amazonaws.com"
                ]
            },
            "Action": "sts:AssumeRole",
            "Condition": {
                "StringEquals": {
                "aws:SourceAccount": "123456789012"
                },
                "ArnEquals": {
                "aws:SourceArn": "arn:aws:iotevents:us-east-1:123456789012:*"
                }
            }
        }
    ]
}
```

Probleme mit AWS IoT Events Identität und Zugriff beheben

Verwenden Sie die folgenden Informationen, um häufig auftretende Probleme zu diagnostizieren und zu beheben, die bei der Arbeit mit AWS IoT Events und IAM auftreten können.

Themen

- Ich bin nicht berechtigt, eine Aktion durchzuführen in AWS IoT Events
- Ich bin nicht zur Ausführung von iam:PassRole autorisiert.
- Ich möchte Personen außerhalb von mir den Zugriff AWS-Konto auf meine AWS IoT Events Ressourcen ermöglichen

Ich bin nicht berechtigt, eine Aktion durchzuführen in AWS IoT Events

Wenn Ihnen AWS-Managementkonsole mitgeteilt wird, dass Sie nicht berechtigt sind, eine Aktion durchzuführen, müssen Sie sich an Ihren Administrator wenden, um Unterstützung zu erhalten. Ihr Administrator ist die Person, die Ihnen Ihren Benutzernamen und Ihr Passwort zur Verfügung gestellt hat.

Der folgende Beispielfehler tritt auf, wenn der IAM-Benutzer `mateojackson` versucht, die Konsole zum Anzeigen von Details zu einem *`input`* zu verwenden, jedoch nicht über `iotevents:`*`ListInputs`*-Berechtigungen verfügt.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 iotevents:ListInputs on resource: my-example-input
```

In diesem Fall bittet Mateo seinen Administrator um die Aktualisierung seiner Richtlinien, um unter Verwendung der Aktion *`my-example-input`* auf die Ressource `iotevents:`*`ListInput`* zugreifen zu können.

Ich bin nicht zur Ausführung von **`iam:PassRole`** autorisiert.

Wenn Sie die Fehlermeldung erhalten, dass Sie nicht zum Durchführen der `iam:PassRole`-Aktion autorisiert sind, müssen Ihre Richtlinien aktualisiert werden, um eine Rolle an AWS IoT Eventsübergeben zu können.

Einige AWS-Services ermöglichen es Ihnen, eine bestehende Rolle an diesen Dienst zu übergeben, anstatt eine neue Servicerolle oder eine dienstverknüpfte Rolle zu erstellen. Hierzu benötigen Sie Berechtigungen für die Übergabe der Rolle an den Dienst.

Der folgende Beispielfehler tritt auf, wenn ein IAM-Benutzer mit dem Namen `marymajor` versucht, die Konsole zu verwenden, um eine Aktion in AWS IoT Events auszuführen. Die Aktion erfordert jedoch, dass der Service über Berechtigungen verfügt, die durch eine Servicerolle gewährt werden. Mary besitzt keine Berechtigungen für die Übergabe der Rolle an den Dienst.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

In diesem Fall müssen die Richtlinien von Mary aktualisiert werden, um die Aktion `iam:PassRole` ausführen zu können.

Wenn Sie Hilfe benötigen, wenden Sie sich an Ihren AWS Administrator. Ihr Administrator hat Ihnen Ihre Anmeldeinformationen zur Verfügung gestellt.

Ich möchte Personen außerhalb von mir den Zugriff AWS-Konto auf meine AWS IoT Events Ressourcen ermöglichen

Sie können eine Rolle erstellen, mit der Benutzer in anderen Konten oder Personen außerhalb Ihrer Organisation auf Ihre Ressourcen zugreifen können. Sie können festlegen, wem die Übernahme der Rolle anvertraut wird. Für Dienste, die ressourcenbasierte Richtlinien oder Zugriffskontrolllisten (ACLs) unterstützen, können Sie diese Richtlinien verwenden, um Personen Zugriff auf Ihre Ressourcen zu gewähren.

Sehen Sie sich die folgenden Themen an, um herauszufinden, welche Optionen am besten geeignet sind:

- Informationen darüber, ob diese Funktionen AWS IoT Events unterstützt werden, finden Sie unterWie AWS IoT Events funktioniert mit IAM.
- Informationen dazu, wie Sie Zugriff auf Ihre Ressourcen gewähren können, AWS-Konten die Ihnen gehören, finden Sie im IAM-Benutzerhandbuch unter Gewähren des Zugriffs auf einen IAM-Benutzer in einem anderen AWS-Konto , den Sie besitzen.
- Informationen dazu, wie Sie Dritten Zugriff auf Ihre Ressourcen gewähren können AWS-Konten, finden Sie AWS-Konten im IAM-Benutzerhandbuch unter Gewähren des Zugriffs für Dritte.
- Informationen dazu, wie Sie über einen Identitätsverbund Zugriff gewähren, finden Sie unter Gewähren von Zugriff für extern authentifizierte Benutzer (Identitätsverbund) im IAM-Benutzerhandbuch.
- Informationen zum Unterschied zwischen der Verwendung von Rollen und ressourcenbasierten Richtlinien für den kontoübergreifenden Zugriff finden Sie unter Kontoübergreifender Ressourcenzugriff in IAM im IAM-Benutzerhandbuch.

Überwachung AWS IoT Events zur Aufrechterhaltung von Zuverlässigkeit, Verfügbarkeit und Leistung

Die Überwachung ist ein wichtiger Bestandteil der Aufrechterhaltung der Zuverlässigkeit, Verfügbarkeit AWS IoT Events und Leistung Ihrer AWS Lösungen. Sie sollten Überwachungsdaten aus allen Teilen Ihrer AWS Lösung sammeln, damit Sie einen etwaigen Ausfall an mehreren Stellen

leichter debuggen können. Bevor Sie mit der Überwachung beginnen AWS IoT Events, sollten Sie einen Überwachungsplan erstellen, der Antworten auf die folgenden Fragen enthält:

- Was sind Ihre Überwachungsziele?
- Welche Ressourcen möchten Sie überwachen?
- Wie oft werden diese Ressourcen überwacht?
- Welche Überwachungs-Tools möchten Sie verwenden?
- Wer soll die Überwachungsaufgaben ausführen?
- Wer soll benachrichtigt werden, wenn Fehler auftreten?

Der nächste Schritt besteht darin, eine Ausgangsbasis für die normale AWS IoT Events Leistung in Ihrer Umgebung festzulegen, indem Sie die Leistung zu verschiedenen Zeiten und unter verschiedenen Lastbedingungen messen. Speichern Sie bei der Überwachung von AWS IoT Events historische Überwachungsdaten, damit Sie diese mit aktuellen Leistungsdaten vergleichen, normale Leistungsmuster bestimmen, Leistungsprobleme erkennen und Methoden zur Fehlerbehebung ableiten können.

Wenn Sie beispielsweise Amazon EC2 verwenden, können Sie die CPU-Auslastung und die Festplatte I/O, and network utilization for your instances. When performance falls outside your established baseline, you might need to reconfigure or optimize the instance to reduce CPU utilization, improve disk I/O überwachen oder den Netzwerkverkehr reduzieren.

Topics

Verfügbare Tools zur Überwachung AWS IoT Events

AWS bietet verschiedene Tools, die Sie zur Überwachung verwenden können AWS IoT Events. Sie können einige dieser Tools so konfigurieren, dass diese die Überwachung für Sie übernehmen, während bei anderen Tools ein manuelles Eingreifen nötig ist. Wir empfehlen, dass Sie die Überwachungsaufgaben möglichst automatisieren.

Automatisierte Überwachungstools

Sie können die folgenden automatisierten Überwachungstools verwenden, um zu beobachten AWS IoT Events und zu melden, wenn etwas nicht stimmt:

- Amazon CloudWatch Logs — Überwachen, speichern und greifen Sie auf Ihre Protokolldateien aus AWS CloudTrail oder anderen Quellen zu. Weitere Informationen finden Sie unter Verwenden von CloudWatch Amazon-Dashboards im CloudWatch Amazon-Benutzerhandbuch.
- AWS CloudTrail Protokollüberwachung — Teilen Sie Protokolldateien zwischen Konten, überwachen CloudTrail Sie Protokolldateien in Echtzeit, indem Sie sie an CloudWatch Logs senden, schreiben Sie Protokollverarbeitungsanwendungen in Java und überprüfen Sie, ob sich Ihre Protokolldateien nach der Lieferung von nicht geändert haben. CloudTrail Weitere Informationen finden Sie im AWS CloudTrail Benutzerhandbuch unter Arbeiten mit CloudTrail Protokolldateien.

Manuelle Überwachungstools

Ein weiterer wichtiger Teil der Überwachung AWS IoT Events umfasst die manuelle Überwachung der Elemente, die von den CloudWatch Alarmen nicht abgedeckt werden. Die Konsolen-Dashboards AWS IoT Events CloudWatch, und andere AWS Konsolen-Dashboards bieten einen at-a-glance Überblick über den Zustand Ihrer AWS Umgebung. Wir empfehlen, dass Sie auch die Protokolldateien unter AWS IoT Eventsüberprüfen.

- Die AWS IoT Events Konsole zeigt:
  - Detektormodelle
  - Detektoren
  - Eingaben
  - Einstellungen
- Auf der CloudWatch Startseite wird Folgendes angezeigt:
  - Aktuelle Alarme und Status
  - Diagramme mit Alarmen und Ressourcen
  - Servicestatus

  Darüber hinaus können CloudWatch Sie Folgendes verwenden:

  - Erstellen Erstellen Sie ein CloudWatch Dashboard zur Überwachung der Dienste, die Ihnen wichtig sind

- Aufzeichnen von Metrikdaten, um Probleme zu beheben und Trends zu erkennen
- Suchen und durchsuchen Sie alle Ihre AWS Ressourcenmetriken
- Erstellen und Bearbeiten von Alarmen, um über Probleme benachrichtigt zu werden

Überwachung AWS IoT Events mit Amazon CloudWatch

Wenn Sie ein AWS IoT Events Detektormodell entwickeln oder debuggen, müssen Sie wissen, was AWS IoT Events gerade passiert und welche Fehler dabei auftreten. Amazon CloudWatch überwacht Ihre AWS Ressourcen und die Anwendungen, auf denen Sie laufen, AWS in Echtzeit. Damit CloudWatch erhalten Sie systemweiten Einblick in die Ressourcennutzung, die Anwendungsleistung und den Betriebszustand. Aktivieren Sie die CloudWatch Amazon-Protokollierung bei der Entwicklung von AWS IoT Events Detektormodellenenthält Informationen zur Aktivierung der CloudWatch Protokollierung für AWS IoT Events. Um Protokolle wie das unten gezeigte zu generieren, müssen Sie den Ausführlichkeitsgrad auf „Debug“ setzen und ein oder mehrere Debug-Ziele angeben, d. h. einen Modellnamen des Detektors und ein optionales. KeyValue

Das folgende Beispiel zeigt einen Protokolleintrag auf CloudWatch DEBUG-Ebene, der von generiert wurde. AWS IoT Events

```
{
  "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
    }
  ]
}
```

Protokollieren von AWS IoT Events API-Aufrufen mit AWS CloudTrail

AWS IoT Events ist in einen Dienst integriert AWS CloudTrail, der eine Aufzeichnung der Aktionen bereitstellt, die von einem Benutzer, einer Rolle oder einem AWS Dienst in ausgeführt wurden AWS IoT Events. CloudTrail erfasst alle API-Aufrufe AWS IoT Events als Ereignisse, einschließlich Aufrufe von der AWS IoT Events Konsole und von Codeaufrufen an die AWS IoT Events APIs.

Wenn Sie einen Trail erstellen, können Sie die kontinuierliche Bereitstellung von CloudTrail Ereignissen an einen Amazon S3 S3-Bucket aktivieren, einschließlich Ereignissen für AWS IoT Events. Wenn Sie keinen Trail konfigurieren, können Sie die neuesten Ereignisse trotzdem in der CloudTrail Konsole im Ereignisverlauf anzeigen. Anhand der von gesammelten Informationen können Sie die Anfrage ermitteln CloudTrail, an die die Anfrage gestellt wurde AWS IoT Events, die IP-Adresse, von der aus die Anfrage gestellt wurde, wer die Anfrage gestellt hat, wann sie gestellt wurde, und weitere Details.

Weitere Informationen CloudTrail dazu finden Sie im AWS CloudTrail Benutzerhandbuch.

AWS IoT Events Informationen in CloudTrail

CloudTrail ist in Ihrem AWS Konto aktiviert, wenn Sie das Konto erstellen. Wenn eine Aktivität in stattfindet AWS IoT Events, wird diese Aktivität zusammen mit anderen CloudTrail AWS Serviceereignissen in der Ereignishistorie aufgezeichnet. Sie können aktuelle Ereignisse in Ihrem AWS Konto ansehen, suchen und herunterladen. Weitere Informationen finden Sie unter Arbeiten mit dem CloudTrail Ereignisverlauf.

Für eine fortlaufende Aufzeichnung der Ereignisse in Ihrem AWS Konto, einschließlich der Ereignisse für AWS IoT Events, erstellen Sie einen Trail. Ein Trail ermöglicht CloudTrail die Übermittlung von Protokolldateien an einen Amazon S3 S3-Bucket. Wenn Sie einen Trail in der Konsole erstellen, gilt der Trail standardmäßig für alle AWS Regionen. Der Trail protokolliert Ereignisse aus allen Regionen der AWS Partition und übermittelt die Protokolldateien an den von Ihnen angegebenen Amazon S3 S3-Bucket. Darüber hinaus können Sie andere AWS Dienste konfigurieren, um die in den CloudTrail Protokollen gesammelten Ereignisdaten weiter zu analysieren und darauf zu reagieren. Weitere Informationen finden Sie unter:

- Einen Trail für dein AWS Konto erstellen
- In CloudTrail unterstützte Services und Integrationen
- Konfiguration von Amazon SNS SNS-Benachrichtigungen für CloudTrail
- Empfangen von CloudTrail Protokolldateien aus mehreren Regionen und Empfangen von CloudTrail Protokolldateien von mehreren Konten

Jeder Ereignis- oder Protokolleintrag enthält Informationen zu dem Benutzer, der die Anforderung generiert hat. Die Identitätsinformationen unterstützen Sie bei der Ermittlung der folgenden Punkte:

- Gibt an, ob die Anforderung mit Root- oder IAM-Benutzer-Anmeldeinformationen ausgeführt wurde.
- Gibt an, ob die Anforderung mit temporären Sicherheitsanmeldeinformationen für eine Rolle oder einen Verbundbenutzer gesendet wurde.
- Ob die Anfrage von einem anderen AWS Dienst gestellt wurde.

Weitere Informationen finden Sie unter dem CloudTrailUserIdentity-Element. AWS IoT Events Aktionen sind in der API-Referenz dokumentiert.AWS IoT Events

AWS IoT Events Logdateieinträge verstehen

Ein Trail ist eine Konfiguration, die die Übertragung von Ereignissen als Protokolldateien an einen von Ihnen angegebenen Amazon S3 S3-Bucket ermöglicht. AWS CloudTrail Protokolldateien enthalten einen oder mehrere Protokolleinträge. Ein Ereignis stellt eine einzelne Anforderung aus einer beliebigen Quelle dar und enthält Informationen über die angeforderte Aktion, Datum und Uhrzeit der Aktion, Anforderungsparameter usw. CloudTrail Protokolldateien sind kein geordneter Stack-Trace der öffentlichen API-Aufrufe, sodass sie nicht in einer bestimmten Reihenfolge angezeigt werden.

Wenn die CloudTrail Protokollierung in Ihrem AWS Konto aktiviert ist, werden die meisten API-Aufrufe von AWS IoT Events Aktionen in CloudTrail Protokolldateien aufgezeichnet, wo sie zusammen mit anderen AWS Serviceaufzeichnungen geschrieben werden. CloudTrailbestimmt anhand eines Zeitraums und der Dateigröße, wann eine neue Datei erstellt und in sie geschrieben werden soll.

Jeder Protokolleintrag enthält Informationen über den Ersteller der Anforderung. Der Benutzeridentität im Protokolleintrag können Sie folgende Informationen entnehmen:

- Gibt an, ob die Anforderung mit Root- oder IAM-Benutzer-Anmeldeinformationen ausgeführt wurde.
- Gibt an, ob die Anforderung mit temporären Sicherheitsanmeldeinformationen für eine Rolle oder einen Verbundbenutzer gesendet wurde.
- Ob die Anfrage von einem anderen AWS Dienst gestellt wurde.

Sie können Ihre Protokolldateien so lange in Ihrem Amazon S3 S3-Bucket speichern, wie Sie möchten, aber Sie können auch Amazon S3 S3-Lebenszyklusregeln definieren, um Protokolldateien automatisch zu archivieren oder zu löschen. Standardmäßig werden die Protokolldateien mit serverseitiger Amazon-S3-Verschlüsselung (SSE) verschlüsselt.

Um bei der Übermittlung der Protokolldatei benachrichtigt zu werden, können Sie so konfigurieren, CloudTrail dass Amazon SNS SNS-Benachrichtigungen veröffentlicht werden, wenn neue Protokolldateien zugestellt werden. Weitere Informationen finden Sie unter Konfigurieren von Amazon SNS-Benachrichtigungen für CloudTrail.

Sie können auch AWS IoT Events Protokolldateien aus mehreren AWS Regionen und mehreren AWS Konten in einem einzigen Amazon S3 S3-Bucket zusammenfassen.

Weitere Informationen finden Sie unter Empfangen von CloudTrail Protokolldateien aus mehreren Regionen und Empfangen von CloudTrail Protokolldateien von mehreren Konten.

Beispiel: DescribeDetector Aktion für CloudTrail

Das folgende Beispiel zeigt einen CloudTrail Protokolleintrag, der die `DescribeDetector` Aktion demonstriert.


```
{
  "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"
}
```

Beispiel: CreateDetectorModel Aktion für CloudTrail

Das folgende Beispiel zeigt einen CloudTrail Protokolleintrag, der die `CreateDetectorModel` Aktion demonstriert.


```
{
  "eventVersion": "1.05",
  "userIdentity": {
    "type": "AssumedRole",
    "principalId": "AKIAI44QH8DHBEXAMPLE:IotEvents-Lambda",
    "arn": "arn:aws:sts::123456789012:assumed-role/IotEvents-RoleForIotEvents-
ABC123DEF456/IotEvents-Lambda",
    "accountId": "123456789012",
    "accessKeyId": "AKIAI44QH8DHBEXAMPLE",
    "sessionContext": {
      "attributes": {
        "mfaAuthenticated": "false",
        "creationDate": "2019-02-07T22:22:30Z"
      },
      "sessionIssuer": {
        "type": "Role",
        "principalId": "AKIAI44QH8DHBEXAMPLE",
        "arn": "arn:aws:iam::123456789012:role/IotEventsLambda-RoleForIotEvents-
ABC123DEF456",
        "accountId": "123456789012",
        "userName": "IotEventsLambda-RoleForIotEvents-ABC123DEF456"
      }
    }
  },
  "eventTime": "2019-02-07T23:54:43Z",
  "eventSource": "iotevents.amazonaws.com",
  "eventName": "CreateDetectorModel",
  "awsRegion": "us-east-1",
  "sourceIPAddress": "192.168.0.1",
  "userAgent": "aws-internal/3",
  "requestParameters": {
    "detectorModelName": "myDetectorModel",
    "key": "HIDDEN_DUE_TO_SECURITY_REASONS",
    "roleArn": "arn:aws:iam::123456789012:role/events_action_execution_role"
  },
  "responseElements": null,
  "requestID": "cecfbfa1-e452-4fa6-b86b-89a89f392b66",
  "eventID": "8138d46b-50a3-4af0-9c5e-5af5ef75ea55",
  "eventType": "AwsApiCall",
```

```
  "recipientAccountId": "123456789012"
}
```

Beispiel: CreateInput Aktion für CloudTrail

Das folgende Beispiel zeigt einen CloudTrail Protokolleintrag, der die `CreateInput` Aktion demonstriert.

```
{
  "eventVersion": "1.05",
  "userIdentity": {
    "type": "AssumedRole",
    "principalId": "AKIAI44QH8DHBEXAMPLE:IotEvents-Lambda",
    "arn": "arn:aws:sts::123456789012:assumed-role/IotEventsLambda-RoleForIotEvents-
ABC123DEF456/IotEvents-Lambda",
    "accountId": "123456789012",
    "accessKeyId": "AKIAI44QH8DHBEXAMPLE",
    "sessionContext": {
      "attributes": {
        "mfaAuthenticated": "false",
        "creationDate": "2019-02-07T22:22:30Z"
      },
      "sessionIssuer": {
        "type": "Role",
        "principalId": "AKIAI44QH8DHBEXAMPLE",
        "arn": "arn:aws:iam::123456789012:role/IotEventsLambda-RoleForIotEvents-
ABC123DEF456",
        "accountId": "123456789012",
        "userName": "IotEventsLambda-RoleForIotEvents-ABC123DEF456"
      }
    }
  },
  "eventTime": "2019-02-07T23:54:43Z",
  "eventSource": "iotevents.amazonaws.com",
  "eventName": "CreateInput",
  "awsRegion": "us-east-1",
  "sourceIPAddress": "192.168.0.1",
  "userAgent": "aws-internal/3",
  "requestParameters": {
    "inputName": "batchputmessagedetectorupdated",
    "inputDescription": "batchputmessagedetectorupdated"
  },
  "responseElements": null,
```

```
  "requestID": "fb315af4-39e9-4114-94d1-89c9183394c1",
  "eventID": "6d8cf67b-2a03-46e6-bbff-e113a7bded1e",
  "eventType": "AwsApiCall",
  "recipientAccountId": "123456789012"
}
```

Beispiel: DeleteDetectorModel Aktion für CloudTrail

Das folgende Beispiel zeigt einen CloudTrail Protokolleintrag, der die `DeleteDetectorModel` Aktion demonstriert.

```
{
  "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"
}
```

Beispiel: DeleteInput Aktion für CloudTrail

Das folgende Beispiel zeigt einen CloudTrail Protokolleintrag, der die `DeleteInput` Aktion demonstriert.

```
{
  "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"
}
```

Beispiel: DescribeDetectorModel Aktion für CloudTrail

Das folgende Beispiel zeigt einen CloudTrail Protokolleintrag, der die `DescribeDetectorModel` Aktion demonstriert.

```
{
  "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"
}
```

Beispiel: DescribeInput Aktion für CloudTrail

Das folgende Beispiel zeigt einen CloudTrail Protokolleintrag, der die `DescribeInput` Aktion demonstriert.

```
{
  "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"
}
```

Beispiel: DescribeLoggingOptions Aktion für CloudTrail

Das folgende Beispiel zeigt einen CloudTrail Protokolleintrag, der die `DescribeLoggingOptions` Aktion demonstriert.

```
{
  "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"
}
```

Beispiel: ListDetectorModels Aktion für CloudTrail

Das folgende Beispiel zeigt einen CloudTrail Protokolleintrag, der die `ListDetectorModels` Aktion demonstriert.

```
{
  "eventVersion": "1.05",
  "userIdentity": {
    "type": "AssumedRole",
    "principalId": "AKIAI44QH8DHBEXAMPLE:IotEvents-EventsLambda",
    "arn": "arn:aws:sts::123456789012:assumed-role/IotEventsLambda-RoleForIotEvents-
ABCD123DEF456/IotEvents-EventsLambda",
    "accountId": "123456789012",
    "accessKeyId": "AKIAI44QH8DHBEXAMPLE",
    "sessionContext": {
      "attributes": {
        "mfaAuthenticated": "false",
        "creationDate": "2019-02-07T22:22:30Z"
      },
      "sessionIssuer": {
        "type": "Role",
        "principalId": "AKIAI44QH8DHBEXAMPLE",
        "arn": "arn:aws:iam::123456789012:role/IotEventsLambda-RoleForIotEvents-
ABCD123DEF456",
        "accountId": "123456789012",
        "userName": "IotEventsLambda-RoleForIotEvents-ABCD123DEF456"
```

```
      }
    }
  },
  "eventTime": "2019-02-07T23:53:23Z",
  "eventSource": "iotevents.amazonaws.com",
  "eventName": "ListDetectorModels",
  "awsRegion": "us-east-1",
  "sourceIPAddress": "192.168.0.1",
  "userAgent": "aws-internal/3",
  "requestParameters": {
    "nextToken": "CkZEZXRlY3Rvck1vZGVsMl9saXN0ZGV0ZWN0b3Jtb2RlbHN0ZXN0X2VlOWJkZTk1YT",
    "maxResults": 3
  },
  "responseElements": null,
  "requestID": "6d70f262-da95-4bb5-94b4-c08369df75bb",
  "eventID": "2d01a25c-d5c7-4233-99fe-ce1b8ec05516",
  "eventType": "AwsApiCall",
  "recipientAccountId": "123456789012"
}
```

Beispiel: ListDetectorModelVersions Aktion für CloudTrail

Das folgende Beispiel zeigt einen CloudTrail Protokolleintrag, der die `ListDetectorModelVersions` Aktion demonstriert.

```
{
  "eventVersion": "1.05",
  "userIdentity": {
    "type": "AssumedRole",
    "principalId": "AKIAI44QH8DHBEXAMPLE:IotEvents-EventsLambda",
    "arn": "arn:aws:sts::123456789012:assumed-role/IotEventsLambda-RoleForIotEvents-
ABCD123DEF456/IotEvents-EventsLambda",
    "accountId": "123456789012",
    "accessKeyId": "AKIAI44QH8DHBEXAMPLE",
    "sessionContext": {
      "attributes": {
        "mfaAuthenticated": "false",
        "creationDate": "2019-02-07T22:22:30Z"
      },
      "sessionIssuer": {
        "type": "Role",
        "principalId": "AKIAI44QH8DHBEXAMPLE",
```

```
        "arn": "arn:aws:iam::123456789012:role/IotEventsLambda-RoleForIotEvents-
ABCD123DEF456",
        "accountId": "123456789012",
        "userName": "IotEventsLambda-RoleForIotEvents-ABCD123DEF456"
      }
    }
  },
  "eventTime": "2019-02-07T23:53:33Z",
  "eventSource": "iotevents.amazonaws.com",
  "eventName": "ListDetectorModelVersions",
  "awsRegion": "us-east-1",
  "sourceIPAddress": "192.168.0.1",
  "userAgent": "aws-internal/3",
  "requestParameters": {
    "detectorModelName": "myDetectorModel",
    "maxResults": 2
  },
  "responseElements": null,
  "requestID": "ebecb277-6bd8-44ea-8abd-fbf40ac044ee",
  "eventID": "fc6281a2-3fac-4e1e-98e0-ca6560b8b8be",
  "eventType": "AwsApiCall",
  "recipientAccountId": "123456789012"
}
```


Beispiel: ListDetectors Aktion für CloudTrail

Das folgende Beispiel zeigt einen CloudTrail Protokolleintrag, der die `ListDetectors` Aktion demonstriert.


```
{
  "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"
}
```

Beispiel: ListInputs Aktion für CloudTrail

Das folgende Beispiel zeigt einen CloudTrail Protokolleintrag, der die `ListInputs` Aktion demonstriert.

```
{
  "eventVersion": "1.05",
  "userIdentity": {
    "type": "AssumedRole",
    "principalId": "AKIAI44QH8DHBEXAMPLE:IotEvents-EventsLambda",
    "arn": "arn:aws:sts::123456789012:assumed-role/IotEventsLambda-RoleForIotEvents-
ABCD123DEF456/IotEvents-EventsLambda",
    "accountId": "123456789012",
    "accessKeyId": "AKIAI44QH8DHBEXAMPLE",
    "sessionContext": {
      "attributes": {
```

```
        "mfaAuthenticated": "false",
        "creationDate": "2019-02-07T22:22:30Z"
      },
      "sessionIssuer": {
        "type": "Role",
        "principalId": "AKIAI44QH8DHBEXAMPLE",
        "arn": "arn:aws:iam::123456789012:role/IotEventsLambda-RoleForIotEvents-
ABCD123DEF456",
        "accountId": "123456789012",
        "userName": "IotEventsLambda-RoleForIotEvents-ABCD123DEF456"
      }
    }
  },
  "eventTime": "2019-02-07T23:53:57Z",
  "eventSource": "iotevents.amazonaws.com",
  "eventName": "ListInputs",
  "awsRegion": "us-east-1",
  "sourceIPAddress": "192.168.0.1",
  "userAgent": "aws-internal/3",
  "requestParameters": {
    "nextToken": "CkhjYW5hcnlfdGVzdF9pbnB1dF9saXN0ZGV0ZWN0b3Jtb2RlbHN0ZXN0ZDU3OGZ",
    "maxResults": 3
  },
  "responseElements": null,
  "requestID": "dd6762a1-1f24-4e63-a986-5ea3938a03da",
  "eventID": "c500f6d8-e271-4366-8f20-da4413752469",
  "eventType": "AwsApiCall",
  "recipientAccountId": "123456789012"
}
```

Beispiel: PutLoggingOptions Aktion für CloudTrail

Das folgende Beispiel zeigt einen CloudTrail Protokolleintrag, der die `PutLoggingOptions` Aktion demonstriert.

```
{
  "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"
}
```

Beispiel: UpdateDetectorModel Aktion für CloudTrail

Das folgende Beispiel zeigt einen CloudTrail Protokolleintrag, der die `UpdateDetectorModel` Aktion demonstriert.

```
{
  "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"
}
```


Beispiel: UpdateInput Aktion für CloudTrail

Das folgende Beispiel zeigt einen CloudTrail Protokolleintrag, der die `UpdateInput` Aktion demonstriert.

```
{
  "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"
}
```

Beispiel: BatchPutMessage Aktion für CloudTrail

AWS IoT Events kann eine CloudTrail Integration für die API-Protokollierung auf Datenebene verwenden. In diesem Beispiel werden Details zu Datenereignissen durch die `BatchPutMessage` Aktion hinzugefügt.


```
{
            "eventVersion": "1.09",
            "userIdentity": {
                "type": "AssumedRole",
                "principalId": "AKIAI44QH8DHBEXAMPLE:PrincipalId",
                "arn": "arn:aws:sts::123456789012:assumed-role/my-iam-role/my-iam-role-
entity",
                "accountId": "123456789012",
                "accessKeyId": "AKIAI44QH8DHBEXAMPLE",
                "sessionContext": {
                    "sessionIssuer": {
                        "type": "Role",
                        "principalId": "AKIAI44QH8DHBEXAMPLE",
                        "arn": "arn:aws:iam::123456789012:role/my-iam-role",
                        "accountId": "123456789012",
                        "userName": "sample_user_name"
                    },
                    "attributes": {
                        "creationDate": "2024-11-22T18:32:41Z",
                        "mfaAuthenticated": "false"
                    }
                }
            },
            "eventTime": "2024-11-22T18:57:35Z",
            "eventSource": "iotevents.amazonaws.com",
            "eventName": "BatchPutMessage",
            "awsRegion": "us-east-1",
            "sourceIPAddress": "3.239.107.128",
            "userAgent": "aws-internal/3",
            "requestParameters": {
                "messages": [
                    {
                        "messageId": "e306d827-b2e4-4439-9c86-411d4242a397",
                        "payload": "HIDDEN_DUE_TO_SECURITY_REASONS",
                        "inputName": "my_input_name"
                    }
                ]
```

```
            },
            "responseElements": {
                "batchPutMessageErrorEntries": []
            },
            "requestID": "cefc6b63-9ccf-4e31-9177-4aec8e701bfe",
            "eventID": "b994b52c-6011-4e3c-ad5f-e784e732fde0",
            "readOnly": false,
            "resources": [
                {
                    "accountId": "123456789012",
                    "type": "AWS::IoTEvents::Input",
                    "ARN": "arn:aws:iotevents:us-east-1:123456789012:input/
my_input_name"
                }
            ],
            "eventType": "AwsApiCall",
            "managementEvent": false,
            "recipientAccountId": "123456789012",
            "eventCategory": "Data",
            "tlsDetails": {
                "tlsVersion": "TLSv1.3",
                "cipherSuite": "TLS_AES_128_GCM_SHA256",
                "clientProvidedHostHeader": "iotevents.us-east-1.amazonaws.com"
            }
        },
```


Konformitätsvalidierung für AWS IoT Events

Informationen darüber, ob AWS-Service ein AWS-Services in den Geltungsbereich bestimmter Compliance-Programme fällt, finden Sie unter Umfang nach Compliance-Programm AWS-Services unter . Wählen Sie dort das Compliance-Programm aus, an dem Sie interessiert sind. Allgemeine Informationen finden Sie unter AWS Compliance-Programme AWS .

Sie können Prüfberichte von Drittanbietern unter herunterladen AWS Artifact. Weitere Informationen finden Sie unter Berichte herunterladen unter .

Ihre Verantwortung für die Einhaltung der Vorschriften bei der Nutzung AWS-Services hängt von der Vertraulichkeit Ihrer Daten, den Compliance-Zielen Ihres Unternehmens und den geltenden Gesetzen und Vorschriften ab. Weitere Informationen zu Ihrer Verantwortung für die Einhaltung der Vorschriften bei der Nutzung AWS-Services finden Sie in der AWS Sicherheitsdokumentation.

Resilienz in AWS IoT Events

Die AWS globale Infrastruktur basiert auf AWS Regionen und Availability Zones. AWS Regionen stellen mehrere physisch getrennte und isolierte Availability Zones bereit, die mit Netzwerken mit geringer Latenz, hohem Durchsatz und hochredundanten Vernetzungen verbunden sind. Mithilfe von Availability Zones können Sie Anwendungen und Datenbanken erstellen und ausführen, die automatisch Failover zwischen Availability Zones ausführen, ohne dass es zu Unterbrechungen kommt. Availability Zones sind besser hoch verfügbar, fehlertoleranter und skalierbarer als herkömmliche Infrastrukturen mit einem oder mehreren Rechenzentren.

Weitere Informationen zu AWS Regionen und Availability Zones finden Sie unter AWS Globale Infrastruktur.

Infrastruktursicherheit in AWS IoT Events

Als verwalteter Dienst AWS IoT Events ist er durch AWS globale Netzwerksicherheit geschützt. Informationen zu AWS Sicherheitsdiensten und zum AWS Schutz der Infrastruktur finden Sie unter AWS Cloud-Sicherheit. Informationen zum Entwerfen Ihrer AWS Umgebung unter Verwendung der bewährten Methoden für die Infrastruktursicherheit finden Sie unter Infrastructure Protection in Security Pillar AWS Well-Architected Framework.

Sie verwenden AWS veröffentlichte API-Aufrufe für den Zugriff AWS IoT Events über das Netzwerk. Kunden müssen Folgendes unterstützen:

- Transport Layer Security (TLS). Wir benötigen TLS 1.2 und empfehlen TLS 1.3.
- Verschlüsselungs-Suiten mit Perfect Forward Secrecy (PFS) wie DHE (Ephemeral Diffie-Hellman) oder ECDHE (Elliptic Curve Ephemeral Diffie-Hellman). Die meisten modernen Systeme wie Java 7 und höher unterstützen diese Modi.

AWS Servicekontingenten für AWS IoT Events Ressourcen

Der Allgemeine AWS-Referenz Leitfaden enthält die Standardkontingente AWS IoT Events für ein AWS Konto. Sofern nicht anders angegeben, gilt jedes Kontingent pro AWS Region. Weitere Informationen finden Sie im Allgemeine AWS-Referenz Handbuch unter AWS IoT Events Endpunkte und Kontingente und AWS Service Quotas.

Um eine Erhöhung des Servicekontingents zu beantragen, reichen Sie in der Support Center-Konsole eine Support-Anfrage ein. Weitere Informationen finden Sie unter Beantragen einer Kontingenterhöhung im Service-Quotas-Benutzerhandbuch.

> **Note**
>
> - Alle Namen für Meldermodelle und Eingänge müssen innerhalb eines Kontos eindeutig sein.
> - Sie können die Namen von Meldermodellen und Eingängen nicht mehr ändern, nachdem sie erstellt wurden.

Verschlagworten Sie Ihre Ressourcen AWS IoT Events

Um Ihnen die Verwaltung und Organisation Ihrer Detektormodelle und Eingänge zu erleichtern, können Sie optional jeder dieser Ressourcen Ihre eigenen Metadaten in Form von Tags zuweisen. In diesem Abschnitt werden Tags und deren Erstellung beschrieben.

Grundlagen zu Tags (Markierungen)

Mithilfe von Tags können Sie Ihre AWS IoT Events Ressourcen auf unterschiedliche Weise kategorisieren, z. B. nach Zweck, Eigentümer oder Umgebung. Dies ist nützlich, wenn Sie viele Ressourcen desselben Typs haben. Sie können eine bestimmte Ressource anhand der ihr zugewiesenen Tags schnell identifizieren.

Jedes Tag besteht aus einem Schlüssel und einem optionalen Wert, die Sie beide selbst definieren können. Sie könnten beispielsweise eine Reihe von Tags für Ihre Eingaben definieren, mit deren Hilfe Sie die Geräte, die diese Eingaben senden, anhand ihres Typs verfolgen können. Wir empfehlen die Erstellung von Tag-Schlüsseln, die die Anforderungen der jeweiligen Ressourcenart erfüllen. Die Verwendung einheitlicher Tag-Schlüssel vereinfacht das Verwalten der -Ressourcen.

Sie können anhand der von Ihnen hinzugefügten oder angewendeten Tags nach Ressourcen suchen und diese filtern, Tags verwenden, um Ihre Kosten zu kategorisieren und nachzuverfolgen, und auch Tags verwenden, um den Zugriff auf Ihre Ressourcen zu steuern, wie unter Verwenden von Tags mit IAM-Richtlinien im AWS IoT Entwicklerhandbuch beschrieben.

Um die Bedienung zu vereinfachen, AWS-Managementkonsole bietet der Tag-Editor im eine zentrale, einheitliche Möglichkeit, Ihre Tags zu erstellen und zu verwalten. Weitere Informationen finden Sie unter Erste Schritte mit dem Tag-Editor im AWS Tagging-Ressourcen- und Tag-Editor-Benutzerhandbuch.

Sie können auch mithilfe der AWS CLI und der AWS IoT Events API mit Tags arbeiten. Sie können Tags bei der Erstellung mit Detektormodellen und Eingaben verknüpfen, indem Sie das `"Tags"` Feld in den folgenden Befehlen verwenden:

- CreateDetectorModel
- CreateInput

Sie können Tags für vorhandene Ressourcen, die das Markieren unterstützen, hinzufügen, ändern oder löschen. Verwenden Sie dazu die folgenden Befehle:

- TagResource
- ListTagsForResource
- UntagResource

Sie können Tag (Markierung)-Schlüssel und -Werte bearbeiten und Tags (Markierungen) jederzeit von einer Ressource entfernen. Sie können den Wert eines Tags (Markierung) zwar auf eine leere Zeichenfolge, jedoch nicht null festlegen. Wenn Sie ein Tag (Markierung) mit demselben Schlüssel wie ein vorhandener Tag (Markierung) für die Ressource hinzufügen, wird der alte Wert mit dem neuen überschrieben. Wenn Sie eine Ressource löschen, werden alle der Ressource zugeordneten Tags ebenfalls gelöscht.

Weitere Informationen finden Sie unter Bewährte Methoden für das Markieren von Ressourcen AWS

Tag-Beschränkungen und -Einschränkungen

Die folgenden grundlegenden Einschränkungen gelten für Tags (Markierungen):

- Maximale Anzahl von Tags (Markierungen) pro Ressource: 50
- Maximale Schlüssellänge — 127 Unicode-Zeichen in UTF-8
- Maximale Wertlänge — 255 Unicode-Zeichen in UTF-8
- Bei Tag-Schlüsseln und -Werten muss die Groß- und Kleinschreibung beachtet werden.
- Verwenden Sie das `"aws:"` Präfix nicht in Ihren Tagnamen oder -Werten, da es für AWS die Verwendung reserviert ist. Sie können keine Tag-Namen oder Werte mit diesem Präfix bearbeiten oder löschen. Tags mit diesem Präfix werden nicht zum Limit für Tags pro Ressource gezählt.
- Wenn Ihr Markierungsschema für mehrere -Services und -Ressourcen verwendet wird, denken Sie daran, dass andere Services möglicherweise Einschränkungen für zulässige Zeichen haben. Im allgemeinen zulässige Zeichen: Buchstaben, Leerzeichen und Zahlen, die in UTF-8 darstellbar sind, sowie die folgenden Sonderzeichen: + - = . _ : / @.

Verwenden von Tags mit IAM-Richtlinien

Sie können Tag-basierte Berechtigungen auf Ressourcenebene in den IAM-Richtlinien anwenden, die Sie für AWS IoT Events -API-Aktionen verwenden. Dies ermöglicht Ihnen eine bessere Kontrolle darüber, welche Ressourcen ein Benutzer erstellen, ändern oder verwenden kann.

Sie können das `Condition`-Element (auch als `Condition`-Block bezeichnet) mit den folgenden Bedingungskontextschlüsseln und Werten in einer IAM-Richtlinie zum Steuern des Benutzerzugriffs (Berechtigungen) basierend auf den Tags einer Ressource verwenden:

- Verwenden Sie `aws:ResourceTag/<tag-key>: <tag-value>`, um Benutzeraktionen für Ressourcen mit bestimmten Tags zuzulassen oder zu verweigern.
- Verwenden Sie `aws:RequestTag/<tag-key>: <tag-value>`, um festzulegen, dass ein bestimmtes Tag verwendet (oder nicht verwendet) wird, wenn Sie eine API-Anfrage stellen, um eine Ressource zu erstellen oder zu ändern, die Tags zulässt.
- Verwenden Sie `aws:TagKeys: [<tag-key>, ...]`, um zu verlangen, dass ein bestimmter Satz von Tag-Schlüsseln verwendet wird (oder nicht), wenn eine API-Anforderung zum Erstellen einer Ressource durchgeführt wird, die Tags zulässt.

> **Note**
>
> Die Bedingungskontextschlüssel und -werte in einer IAM-Richtlinie gelten nur für die AWS IoT Events -Aktionen, bei denen eine Kennung für eine Ressource, die Tags zulässt, ein erforderlicher Parameter ist.

Im AWS Identity and Access Management Benutzerhandbuch finden Sie zusätzliche Informationen zur Verwendung von Tags zur Zugriffssteuerung mithilfe von Tags. Der Abschnitt IAM-JSON-Richtlinienreferenz dieses Handbuchs enthält die detaillierte Syntax sowie Beschreibungen und Beispiele für Elemente, Variablen und die Auswertungslogik von JSON-Richtlinien in IAM.

Die folgende Beispielrichtlinie wendet zwei auf Tags basierende Einschränkungen an. Ein Benutzer, der durch diese Richtlinie eingeschränkt ist:

- Kann keiner Ressource den Tag „env = prod“ zuweisen (im Beispiel vgl. die Zeile `"aws:RequestTag/env" : "prod"`
- Kann keine Ressource modifizieren oder darauf zugreifen, die den Tag „env=prod“ aufweist (im Beispiel vgl. die Zeile `"aws:ResourceTag/env" : "prod"`).

JSON

```
{
```

```
    "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": "*"
        }
    ]
}
```

Sie können auch mehrere Tag-Werte für einen bestimmten Tag-Schlüssel angeben, indem Sie sie wie folgt in eine Liste einschließen.

```
        "StringEquals" : {
          "aws:ResourceTag/env" : ["dev", "test"]
        }
```

> **Note**
>
> Wenn Sie Benutzern den Zugriff zu Ressourcen auf der Grundlage von Tags (Markierungen) gewähren oder verweigern, müssen Sie daran denken, Benutzern explizit das Hinzufügen und Entfernen dieser Tags (Markierungen) von den jeweiligen Ressourcen unmöglich zu machen. Andernfalls können Benutzer möglicherweise Ihre Einschränkungen umgehen und sich Zugriff auf eine Ressource verschaffen, indem sie ihre Tags (Markierungen) modifizieren.

Problembehebung AWS IoT Events

Diese Anleitung zur Fehlerbehebung bietet Lösungen für häufig auftretende Probleme, die bei der Verwendung auftreten können AWS IoT Events. Lesen Sie die Themen, um Probleme bei der Erkennung von Ereignissen, beim Zugriff auf Daten, Berechtigungen, Serviceintegrationen, Gerätekonfigurationen und mehr zu identifizieren und zu lösen. Mit Tipps zur Fehlerbehebung für AWS IoT Events Konsole, API, CLI, Fehler, Latenz und Integrationen soll dieser Leitfaden darauf abzielen, Ihre Probleme schnell zu lösen, damit Sie zuverlässige und skalierbare ereignisgesteuerte Anwendungen erstellen können.

Themen

Allgemeine AWS IoT Events Probleme und Lösungen

Im folgenden Abschnitt finden Sie Informationen zur Behebung von Fehlern und möglichen Lösungen zur Behebung von Problemen mit AWS IoT Events.

Fehler

Fehler bei der Erstellung des Detektormodells

Ich erhalte Fehler, wenn ich versuche, ein Detektormodell zu erstellen.

Lösung

Wenn Sie ein Detektormodell erstellen, müssen Sie die folgenden Einschränkungen berücksichtigen.

- In jedem `action` Feld ist nur eine Aktion zulässig.
- Das `condition` ist erforderlich für`transitionEvents`. Es ist optional für `OnEnter``OnInput`, und `OnExit` Ereignisse.
- Wenn das `condition` Feld leer ist, entspricht das ausgewertete Ergebnis des Bedingungsausdrucks`true`.
- Das ausgewertete Ergebnis des Bedingungsausdrucks sollte ein boolescher Wert sein. Wenn das Ergebnis kein boolescher Wert ist, entspricht es dem `actions` im Ereignis `nextState` angegebenen Wert `false` und löst diesen nicht aus.

Weitere Informationen finden Sie unter AWS IoT Events Einschränkungen und Einschränkungen des Detektormodells.

Aktualisierungen aus einem gelöschten Meldermodell

Ich habe vor ein paar Minuten ein Meldermodell aktualisiert oder gelöscht, erhalte aber immer noch Statusaktualisierungen vom alten Meldermodell über MQTT-Nachrichten oder SNS-Benachrichtigungen.

Lösung

Wenn Sie ein Detektormodell aktualisieren, löschen oder neu erstellen (siehe UpdateDetectorModel), kommt es zu einer Verzögerung, bis alle Detektorinstanzen gelöscht und das neue Modell verwendet wird. Während dieser Zeit werden Eingaben möglicherweise weiterhin von den Instanzen der vorherigen Version des Detektormodells verarbeitet. Möglicherweise erhalten Sie weiterhin Warnmeldungen, die durch das vorherige Meldermodell definiert wurden. Warten Sie mindestens sieben Minuten, bevor Sie das Update erneut überprüfen oder einen Fehler melden.

Fehler beim Auslösen einer Aktion (wenn eine Bedingung erfüllt ist)

Der Detektor kann keine Aktion auslösen oder in einen neuen Zustand übergehen, wenn die Bedingung erfüllt ist.

Lösung

Stellen Sie sicher, dass das ausgewertete Ergebnis des bedingten Ausdrucks des Detektors ein boolescher Wert ist. Wenn das Ergebnis kein boolescher Wert ist, entspricht es dem `action` im Ereignis `nextState` angegebenen Wert `false` und löst diesen nicht aus. Weitere Informationen finden Sie unter Syntax für bedingte Ausdrücke.

Fehler beim Auslösen einer Aktion (bei Überschreitung eines Schwellenwerts)

Der Detektor löst keine Aktion oder keinen Ereignisübergang aus, wenn die Variable in einem bedingten Ausdruck einen bestimmten Wert erreicht.

Lösung

Wenn Sie `setVariable` für`onInput`, `onEnter` oder aktualisieren`onExit`, wird der neue Wert bei der Auswertung `condition` während des aktuellen Verarbeitungszyklus nicht verwendet. Stattdessen wird der ursprüngliche Wert verwendet, bis der aktuelle Zyklus abgeschlossen ist. Sie können dieses Verhalten ändern, indem Sie den `evaluationMethod` Parameter in der Definition des Detektormodells festlegen. Wenn auf gesetzt `evaluationMethod` ist`SERIAL`, werden Variablen aktualisiert und die Ereignisbedingungen in der Reihenfolge ausgewertet, in der die Ereignisse definiert sind. Wenn auf gesetzt `evaluationMethod` ist `BATCH` (Standardeinstellung), werden Variablen aktualisiert und Ereignisse erst ausgeführt, nachdem alle Ereignisbedingungen ausgewertet wurden.

Falsche Verwendung des Status

Der Detektor wechselt in den falschen Status, wenn ich versuche, Nachrichten an Eingänge zu senden, indem `BatchPutMessage` ich

Lösung

Wenn Sie früher BatchPutMessagemehrere Nachrichten an Eingänge senden, ist die Reihenfolge, in der die Nachrichten oder Eingaben verarbeitet werden, nicht garantiert. Um die Bestellung zu garantieren, senden Sie Nachrichten nacheinander und warten Sie jedes Mal`BatchPutMessage`, bis Sie den Erfolg bestätigen.

Verbindungsnachricht

Ich erhalte eine `('Connection aborted.', error(54, 'Connection reset by peer'))` Fehlermeldung, wenn ich versuche, eine API aufzurufen oder aufzurufen.

Lösung

Stellen Sie sicher, dass OpenSSL TLS 1.1 oder eine neuere Version verwendet, um die Verbindung herzustellen. Dies sollte unter den meisten Linux-Distributionen oder Windows Version 7 und höher die Standardeinstellung sein. Benutzer von macOS müssen möglicherweise OpenSSL aktualisieren.

InvalidRequestException Nachricht

Ich erhalte InvalidRequestException , wenn ich versuche anzurufen `CreateDetectorModel` und `UpdateDetectorModel` APIs.

Lösung

Überprüfen Sie Folgendes, um das Problem zu lösen. Weitere Informationen erhalten Sie unter CreateDetectorModel und UpdateDetectorModel.

- Stellen Sie sicher, dass Sie nicht beide `seconds` und `durationExpression` als Parameter von `SetTimerAction` gleichzeitig verwenden.
- Stellen Sie sicher, dass Ihr Zeichenkettenausdruck für gültig `durationExpression` ist. Der Zeichenkettenausdruck kann Zahlen, Variablen (`$variable.<variable-name>`) oder Eingabewerte (`$input.<input-name>.<path-to-datum>`) enthalten.

Amazon CloudWatch **`action.setTimer`** Logs-Fehler

Sie können Amazon CloudWatch Logs einrichten, um AWS IoT Events Detector-Modell-Instances zu überwachen. Im Folgenden sind die häufigsten Fehler aufgeführt AWS IoT Events, die bei der Verwendung von generiert `action.setTimer` werden.

- Fehler: Ihr Dauerausdruck für den angegebenen Timer *`<timer-name>`* konnte nicht zu einer Zahl ausgewertet werden.

Lösung

Stellen Sie sicher, dass Ihr Zeichenkettenausdruck für in eine Zahl umgewandelt werden `durationExpression` kann. Andere Datentypen, wie z. B. Boolean, sind nicht zulässig.

- Fehler: Das ausgewertete Ergebnis Ihres Dauerausdrucks für den angegebenen Timer *<timer-name>* ist größer als 31622440. Um die Genauigkeit zu gewährleisten, stellen Sie sicher, dass sich Ihr Ausdruck für die Dauer auf einen Wert zwischen 60 und 31622400 bezieht.

Lösung

Stellen Sie sicher, dass die Dauer Ihres Timers mindestens 31622400 Sekunden beträgt. Das ausgewertete Ergebnis der Dauer wird auf die nächste ganze Zahl abgerundet.

- Fehler: Das ausgewertete Ergebnis Ihres Ausdrucks für die Dauer für den angegebenen Timer *<timer-name>* ist kleiner als 60. Um die Genauigkeit zu gewährleisten, stellen Sie sicher, dass sich Ihr Ausdruck für die Dauer auf einen Wert zwischen 60 und 31622400 bezieht.

Lösung

Stellen Sie sicher, dass die Dauer Ihres Timers mindestens 60 Sekunden beträgt. Das ausgewertete Ergebnis der Dauer wird auf die nächste ganze Zahl abgerundet.

- Fehler: Ihr Dauerausdruck für den genannten Timer *<timer-name>* konnte nicht ausgewertet werden. Überprüfen Sie die Variablennamen, Eingabenamen und Pfade zu den Daten, um sicherzustellen, dass Sie auf die vorhandenen Variablen und Eingaben verweisen.

Lösung

Stellen Sie sicher, dass sich Ihr Zeichenkettenausdruck auf die vorhandenen Variablen und Eingaben bezieht. Der Zeichenkettenausdruck kann Zahlen, Variablen (`$variable.`*variable-name*) und Eingabewerte (`$input.`*input-name*`.`*path-to-datum*) enthalten.

- Fehler: Der angegebene Timer konnte nicht festgelegt `<timer-name>` werden. Überprüfen Sie den Ausdruck für die Dauer, und versuchen Sie es erneut.

Lösung

Sehen Sie sich die SetTimerActionAktion an, um sicherzustellen, dass Sie die richtigen Parameter angegeben haben, und stellen Sie dann den Timer erneut ein.

Weitere Informationen finden Sie unter CloudWatch Amazon-Protokollierung bei der Entwicklung von AWS IoT Events Meldermodellen aktivieren.

CloudWatch Amazon-Payload-Fehler

Sie können Amazon CloudWatch Logs einrichten, um AWS IoT Events Detector-Modell-Instances zu überwachen. Im Folgenden finden Sie häufig auftretende Fehler und Warnungen AWS IoT Events, die bei der Konfiguration der Aktions-Payload generiert werden.

- Fehler: Wir konnten Ihren Ausdruck für die Aktion nicht auswerten. Stellen Sie sicher, dass sich die Variablennamen, Eingabenamen und Pfade zu den Daten auf die vorhandenen Variablen und Eingabewerte beziehen. Stellen Sie außerdem sicher, dass die Größe der Payload weniger als 1 KB beträgt, was der maximal zulässigen Größe einer Payload entspricht.

  Lösung

  Stellen Sie sicher, dass Sie die richtigen Variablennamen, Eingabenamen und Pfade zu den Daten eingeben. Möglicherweise erhalten Sie diese Fehlermeldung auch, wenn die Aktionsnutzlast größer als 1 KB ist.
- Fehler: Wir konnten Ihren Inhaltsausdruck nicht nach der Payload von analysieren. *<action-type>* Geben Sie einen Inhaltsausdruck mit der richtigen Syntax ein.

  Lösung

  Der Inhaltsausdruck kann Zeichenfolgen (`'`*`string`*`'`), Variablen (`$variable.`*`variable-name`*), Eingabewerte (`$input.`*`input-name.path-to-datum`*), Zeichenkettenverkettungen und Zeichenfolgen enthalten, die Folgendes enthalten: `${}`
- Fehler: Ihr Payload-Ausdruck {*`expression`*} ist nicht gültig. Der definierte Payload-Typ ist JSON. Sie müssen also einen Ausdruck angeben, der AWS IoT Events eine Zeichenfolge ergibt.

  Lösung

  Wenn der angegebene Payload-Typ JSON ist, wird AWS IoT Events zunächst geprüft, ob der Service Ihren Ausdruck als Zeichenfolge auswerten kann. Das ausgewertete Ergebnis darf weder ein Boolescher Wert noch eine Zahl sein. Wenn die Überprüfung fehlschlägt, erhalten Sie möglicherweise diesen Fehler.
- Warnung: Die Aktion wurde ausgeführt, aber wir konnten Ihren Inhaltsausdruck für die Aktionsnutzlast nicht als gültiges JSON auswerten. Der definierte Payload-Typ ist JSON.

Lösung

Stellen Sie sicher, dass Ihr Inhaltsausdruck für die Aktionsnutzlast als gültiges JSON ausgewertet werden AWS IoT Events kann, wenn Sie den Payload-Typ als definieren. `JSON` AWS IoT Events führt die Aktion aus, auch wenn der Inhaltsausdruck nicht als gültiges JSON ausgewertet werden AWS IoT Events kann.

Weitere Informationen finden Sie unter CloudWatch Amazon-Protokollierung bei der Entwicklung von AWS IoT Events Meldermodellen aktivieren.

Inkompatible Datentypen

Meldung: `<reference>` Im folgenden Ausdruck wurden inkompatible Datentypen [`<inferred-types>`] gefunden: `<expression>`

Lösung

Dieser Fehler kann aus einem der folgenden Gründe auftreten:

- Die ausgewerteten Ergebnisse Ihrer Verweise sind nicht mit anderen Operanden in Ihren Ausdrücken kompatibel.
- Der Typ des an eine Funktion übergebenen Arguments wird nicht unterstützt.

Wenn Sie Verweise in Ausdrücken verwenden, überprüfen Sie Folgendes:

- Wenn Sie eine Referenz als Operanden mit einem oder mehreren Operatoren verwenden, stellen Sie sicher, dass alle Datentypen, auf die Sie verweisen, kompatibel sind.

  Im folgenden Ausdruck 2 ist Integer beispielsweise ein Operand sowohl der == Operatoren als auch. && Um sicherzustellen, dass die Operanden kompatibel sind `$variable.testVariable + 1` und auf eine Ganzzahl oder Dezimalzahl verweisen `$variable.testVariable` müssen.

  Außerdem 1 ist Integer ein Operand des Operators+. `$variable.testVariable`Muss daher auf eine Ganzzahl oder Dezimalzahl verweisen.

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

- Wenn Sie eine Referenz als Argument verwenden, das an eine Funktion übergeben wird, stellen Sie sicher, dass die Funktion die Datentypen unterstützt, auf die Sie verweisen.

Für die folgende `timeout("time-name")` Funktion ist beispielsweise eine Zeichenfolge mit doppelten Anführungszeichen als Argument erforderlich. Wenn Sie einen Verweis für den *timer-name* Wert verwenden, müssen Sie auf eine Zeichenfolge mit doppelten Anführungszeichen verweisen.

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

> **Note**
>
> Wenn Sie für die `convert(type, expression)` Funktion eine Referenz für den *type* Wert verwenden, muss das ausgewertete Ergebnis Ihrer Referenz`String`,`Decimal`, oder sein`Boolean`.

Weitere Informationen finden Sie unter AWS IoT Events Referenz für Eingaben und Variablen in Ausdrücken.

Nachricht konnte nicht gesendet werden an AWS IoT Events

Nachricht: Nachricht konnte nicht an Iot Events gesendet werden

Lösung

Dieser Fehler kann aus den folgenden Gründen auftreten:

- Die Nutzlast der Eingabenachricht enthält nicht die`Input attribute Key`.
- Der `Input attribute Key` befindet sich nicht in demselben JSON-Pfad, der in der Eingabedefinition angegeben ist.
- Die Eingabenachricht entspricht nicht dem Schema, wie es in der AWS IoT Events Eingabe definiert ist.

> **Note**
>
> Bei der Datenaufnahme von anderen Diensten wird es ebenfalls zu einem Fehler kommen.

Example

Beispielsweise schlägt die AWS IoT Core AWS IoT Regel mit der folgenden Meldung fehl `Verify the Input Attribute key.`

Um dieses Problem zu lösen, stellen Sie sicher, dass das Eingabe-Payload-Nachrichtenschema der AWS IoT Events Eingabedefinition entspricht und der `Input attribute Key` Speicherort übereinstimmt. Weitere Informationen finden Sie unter, Erstellen Sie eine Eingabe für Modelle in AWS IoT Events um zu erfahren, wie Sie Eingaben definieren AWS IoT Events .

Fehlerbehebung bei einem Detektormodell durch Ausführen von Analysen in AWS IoT Events

AWS IoT Events kann Ihr Detektormodell analysieren und Analyseergebnisse generieren, ohne Eingabedaten an Ihr Detektormodell zu senden. AWS IoT Events führt eine Reihe von Analysen durch, die in diesem Abschnitt beschrieben werden, um Ihr Detektormodell zu überprüfen. Diese erweiterte Lösung zur Fehlerbehebung fasst auch Diagnoseinformationen zusammen, einschließlich Schweregrad und Lokalisation, sodass Sie potenzielle Probleme in Ihrem Meldermodell schnell finden und beheben können. Weitere Informationen zu Diagnosefehlertypen und Meldungen für Ihr Meldermodell finden Sie unterAnalyse von Detektormodellen und Diagnoseinformationen für AWS IoT Events.

Sie können die AWS IoT Events Konsole, API, AWS Command Line Interface (AWS CLI) oder AWS das SDK verwenden, um diagnostische Fehlermeldungen aus der Analyse Ihres Meldermodells anzuzeigen.

> **Note**
>
> - Sie müssen alle Fehler beheben, bevor Sie Ihr Meldermodell veröffentlichen können.
> - Wir empfehlen Ihnen, die Warnungen zu lesen und die erforderlichen Maßnahmen zu ergreifen, bevor Sie Ihr Detektormodell in Produktionsumgebungen verwenden. Andernfalls funktioniert das Meldermodell möglicherweise nicht wie erwartet.
> - Sie können bis zu 10 Analysen gleichzeitig im `RUNNING` Status haben.

Informationen zur Analyse Ihres Detektormodells finden Sie unter Analysieren Sie ein Detektormodell für AWS IoT Events (Konsole) oderAnalysieren Sie ein Detektormodell in AWS IoT Events (AWS CLI).

Topics

Analyse von Detektormodellen und Diagnoseinformationen für AWS IoT Events

Bei Analysen von Detektormodellen werden die folgenden Diagnoseinformationen gesammelt:

- Stufe — Der Schweregrad des Analyseergebnisses. Basierend auf dem Schweregrad lassen sich die Analyseergebnisse in drei allgemeine Kategorien einteilen:
  - Information (`INFO`) — Ein Informationsergebnis gibt Aufschluss über ein bedeutendes Feld in Ihrem Detektormodell. Für diese Art von Ergebnis sind in der Regel keine sofortigen Maßnahmen erforderlich.
  - Warnung (`WARNING`) — Ein Warnergebnis lenkt besondere Aufmerksamkeit auf Felder, die Probleme mit Ihrem Meldermodell verursachen könnten. Wir empfehlen Ihnen, die Warnungen zu lesen und die erforderlichen Maßnahmen zu ergreifen, bevor Sie Ihr Meldermodell in Produktionsumgebungen verwenden. Andernfalls funktioniert das Meldermodell möglicherweise nicht wie erwartet.
  - Fehler (`ERROR`) — Ein Fehlerergebnis informiert Sie über ein Problem in Ihrem Detektormodell. AWS IoT Events führt diesen Satz von Analysen automatisch durch, wenn Sie versuchen, das Detektormodell zu veröffentlichen. Sie müssen alle Fehler beheben, bevor Sie das Detektormodell veröffentlichen können.
- Position — Enthält Informationen, anhand derer Sie das Feld in Ihrem Detektormodell lokalisieren können, auf das sich das Analyseergebnis bezieht. Ein Standort umfasst in der Regel den Namen des Status, den Namen des Übergangsereignisses, den Namen des Ereignisses und den Ausdruck (z. B.`in state TemperatureCheck in onEnter in event Init in action setVariable`).
- Typ — Der Typ des Analyseergebnisses. Analysetypen lassen sich in die folgenden Kategorien einteilen:

  - `supported-actions`— AWS IoT Events kann Aktionen aufrufen, wenn ein bestimmtes Ereignis oder ein Übergangsereignis erkannt wird. Sie können integrierte Aktionen definieren, um einen Timer zu verwenden oder eine Variable festzulegen oder Daten an andere AWS Dienste zu senden. Sie müssen Aktionen angeben, die mit anderen AWS Diensten in einer AWS Region funktionieren, in der die AWS Dienste verfügbar sind.
  - `service-limits`— Dienstkontingente, auch Limits genannt, sind die maximale oder minimale Anzahl von Serviceressourcen oder Vorgängen für Ihr AWS Konto. Sofern nicht anders angegeben, gilt jedes Kontingent spezifisch für eine Region. Je nach Ihren Geschäftsanforderungen können Sie Ihr Meldermodell aktualisieren, um Grenzwerte zu vermeiden, oder eine Erhöhung des Kontingents beantragen. Sie können Erhöhungen für einige Kontingente beantragen und andere Kontingente können nicht erhöht werden. Weitere Informationen finden Sie unter Kontingente.
- **`structure`**— Das Meldermodell muss alle erforderlichen Komponenten wie Zustände enthalten und einer Struktur folgen, die dies AWS IoT Events unterstützt. Ein Detektormodell muss mindestens einen Zustand und eine Bedingung haben, die die eingehenden Eingabedaten auswertet, um signifikante Ereignisse zu erkennen. Wenn ein Ereignis erkannt wird, wechselt das Detektormodell in den nächsten Status und kann Aktionen auslösen. Diese Ereignisse werden als Übergangsereignisse bezeichnet. Ein Übergangsereignis muss den nächsten Status zum Eintritt anweisen.
- **`expression-syntax`**— AWS IoT Events bietet mehrere Möglichkeiten, Werte anzugeben, wenn Sie Detektormodelle erstellen und aktualisieren. Sie können Literale, Operatoren, Funktionen, Referenzen und Substitutionsvorlagen in den Ausdrücken verwenden. Sie können Ausdrücke verwenden, um Literalwerte anzugeben, oder Sie AWS IoT Events können die Ausdrücke auswerten, bevor Sie bestimmte Werte angeben. Ihr Ausdruck muss der erforderlichen Syntax entsprechen. Weitere Informationen finden Sie unter Ausdrücke zum Filtern, Transformieren und Verarbeiten von Ereignisdaten.

  Detector Model-Ausdrücke in AWS IoT Events können auf bestimmte Daten oder eine Ressource verweisen.

  - **`data-type`**— AWS IoT Events unterstützt Integer-, Dezimal-, String- und Boolesche Datentypen. Wenn die Daten eines Datentyps bei der Auswertung von Ausdrücken automatisch in einen anderen konvertiert werden AWS IoT Events können, sind diese Datentypen kompatibel.

> **Note**
>
> - Integer und Decimal sind die einzigen kompatiblen Datentypen, die von unterstützt werden AWS IoT Events.
> - AWS IoT Events kann keine arithmetischen Ausdrücke auswerten, da eine Ganzzahl nicht in eine Zeichenfolge konvertiert werden AWS IoT Events kann.

- **`referenced-data`**— Sie müssen die Daten definieren, auf die in Ihrem Detektormodell verwiesen wird, bevor Sie die Daten verwenden können. Wenn Sie beispielsweise Daten an eine DynamoDB-Tabelle senden möchten, müssen Sie eine Variable definieren, die auf den Tabellennamen verweist, bevor Sie die Variable in einem Ausdruck () `$variable.TableName` verwenden können.
- **`referenced-resource`**— Ressourcen, die das Detektormodell verwendet, müssen verfügbar sein. Sie müssen Ressourcen definieren, bevor Sie sie verwenden können. Sie möchten beispielsweise ein Detektormodell zur Überwachung der Temperatur eines Gewächshauses erstellen. Sie müssen eine Eingabe (`$input.TemperatureInput`) definieren, um eingehende Temperaturdaten an Ihr Meldermodell weiterzuleiten, bevor Sie `$input.TemperatureInput.sensorData.temperature` die Temperatur als Referenz verwenden können.

Im folgenden Abschnitt finden Sie Informationen zur Behebung von Fehlern und zur Suche nach möglichen Lösungen anhand der Analyse Ihres Detektormodells.

Beheben Sie Fehler im Detektormodell in AWS IoT Events

Die oben beschriebenen Fehlertypen liefern Diagnoseinformationen zu einem Detektormodell und entsprechen Meldungen, die Sie möglicherweise abrufen. Verwenden Sie diese Meldungen und Lösungsvorschläge, um Fehler mit Ihrem Meldermodell zu beheben.

Nachrichten und Lösungen

- Location
- supported-actions
- service-limits
- structure
- expression-syntax

- data-type
- referenced-data
- referenced-resource

`Location`

Ein Analyseergebnis mit Informationen über`Location`, entspricht der folgenden Fehlermeldung:

- Meldung — Enthält zusätzliche Informationen zum Analyseergebnis. Dies kann eine Information, Warnung oder Fehlermeldung sein.

  Lösung: Möglicherweise erhalten Sie diese Fehlermeldung, wenn Sie eine Aktion angegeben haben, die AWS IoT Events derzeit nicht unterstützt wird. Eine Liste der unterstützten Aktionen finden Sie unterUnterstützte Aktionen zum Empfangen von Daten und Auslösen von Aktionen in AWS IoT Events.

`supported-actions`

Ein Analyseergebnis mit Informationen über`supported-actions`, entspricht den folgenden Fehlermeldungen:

- Meldung: Ungültiger Aktionstyp in der Aktionsdefinition vorhanden:*`action-definition`*.

  Lösung: Möglicherweise erhalten Sie diese Fehlermeldung, wenn Sie eine Aktion angegeben haben, die AWS IoT Events derzeit nicht unterstützt wird. Eine Liste der unterstützten Aktionen finden Sie unterUnterstützte Aktionen zum Empfangen von Daten und Auslösen von Aktionen in AWS IoT Events.

- Meldung: Die DetectorModel Definition hat eine *`aws-service`* Aktion, aber der *`aws-service`* Dienst wird in der Region nicht unterstützt*`region-name`*.

  Lösung: Möglicherweise erhalten Sie diese Fehlermeldung, wenn die von Ihnen angegebene Aktion von unterstützt wird AWS IoT Events, die Aktion jedoch in Ihrer aktuellen Region nicht verfügbar ist. Dies kann auftreten, wenn Sie versuchen, Daten an einen AWS Dienst zu senden, der in der Region nicht verfügbar ist. Sie müssen außerdem dieselbe Region für beide AWS IoT Events und die AWS Dienste, die Sie verwenden, auswählen.

`service-limits`

Ein Analyseergebnis mit Informationen über`service-limits`, entspricht den folgenden Fehlermeldungen:

- Meldung: Der in der Nutzlast zulässige Inhaltsausdruck hat die *`content-expression-size`* Bytebegrenzung für das Ereignis *`event-name`* im Status *`state-name`* überschritten.

  Lösung: Möglicherweise erhalten Sie diese Fehlermeldung, wenn der Inhaltsausdruck für Ihre Aktionsnutzlast größer als 1024 Byte ist. Die Größe des Inhaltsausdrucks für eine Payload kann bis zu 1024 Byte betragen.
- Meldung: Die Anzahl der in der Definition des Detektormodells zulässigen Zustände hat den Grenzwert *`states-per-detector-model`* überschritten.

  Lösung: Möglicherweise erhalten Sie diese Fehlermeldung, wenn Ihr Detektormodell mehr als 20 Zustände hat. Ein Detektormodell kann bis zu 20 Zustände haben.
- Meldung: Die Dauer des Timers *`timer-name`* sollte mindestens *`minimum-timer-duration`* Sekunden lang sein.

  Lösung: Möglicherweise erhalten Sie diese Fehlermeldung, wenn die Dauer Ihres Timers weniger als 60 Sekunden beträgt. Wir empfehlen, dass die Dauer eines Timers zwischen 60 und 31622400 Sekunden liegt. Wenn Sie einen Ausdruck für die Dauer Ihres Timers angeben, wird das ausgewertete Ergebnis des Dauerausdrucks auf die nächste ganze Zahl abgerundet.
- Meldung: Die Anzahl der pro Ereignis zulässigen Aktionen hat den Grenzwert *`actions-per-event`* in der Definition des Detektormodells überschritten

  Lösung: Möglicherweise erhalten Sie diese Fehlermeldung, wenn das Ereignis mehr als 10 Aktionen umfasst. Sie können bis zu 10 Aktionen für jedes Ereignis in Ihrem Meldermodell einrichten.
- Meldung: Die Anzahl der pro Status zulässigen Übergangsereignisse hat den Grenzwert *`transition-events-per-state`* in der Definition des Detektormodells überschritten.

  Lösung: Möglicherweise erhalten Sie diese Fehlermeldung, wenn der Status mehr als 20 Übergangsereignisse enthält. Sie können bis zu 20 Übergangsereignisse für jeden Status in Ihrem Detektormodell haben.
- Meldung: Die Anzahl der pro Status zulässigen Ereignisse hat den Grenzwert *`events-per-state`* in der Definition des Detektormodells überschritten

Lösung: Möglicherweise erhalten Sie diese Fehlermeldung, wenn der Status mehr als 20 Ereignisse enthält. Sie können bis zu 20 Ereignisse für jeden Status in Ihrem Detektormodell haben.

- Meldung: Die maximale Anzahl von Meldermodellen, die einem einzelnen Eingang zugeordnet werden können, hat möglicherweise den Grenzwert erreicht. *`input-name`*Die Eingabe wird in *`detector-models-per-input`* Detektormodellrouten verwendet.

  Lösung: Möglicherweise erhalten Sie diese Warnmeldung, wenn Sie versuchen, eine Eingabe an mehr als 10 Detektormodelle weiterzuleiten. Einem einzelnen Detektormodell können bis zu 10 verschiedene Detektormodelle zugeordnet werden.

`structure`

Ein Analyseergebnis mit Informationen über`structure`, entspricht den folgenden Fehlermeldungen:

- Meldung: Für Aktionen ist möglicherweise nur ein Typ definiert, es wurde jedoch eine Aktion mit *`number-of-types`* Typen gefunden. Bitte teilen Sie sie in separate Aktionen auf.

  Lösung: Möglicherweise erhalten Sie diese Fehlermeldung, wenn Sie mithilfe von API-Operationen zwei oder mehr Aktionen in einem einzigen Feld angegeben haben, um Ihr Meldermodell zu erstellen oder zu aktualisieren. Sie können eine Reihe von `Action` Objekten definieren. Stellen Sie sicher, dass Sie jede Aktion als separates Objekt definieren.

- Nachricht: Die TransitionEvent *`transition-event-name`* Übergänge in einen Zustand*`state-name`*, der nicht existiert.

  Lösung: Möglicherweise erhalten Sie diese Fehlermeldung, wenn Sie den nächsten Status AWS IoT Events , auf den Ihr Übergangsereignis verwiesen hat, nicht finden können. Vergewissern Sie sich, dass der nächste Status definiert ist und dass Sie den richtigen Statusnamen eingegeben haben.

- Meldung: Der DetectorModelDefinition hatte einen gemeinsamen Statusnamen: Zustand *`state-name`* mit *`number-of-states`* Wiederholungen gefunden.

  Lösung: Möglicherweise erhalten Sie diese Fehlermeldung, wenn Sie denselben Namen für einen oder mehrere Staaten verwenden. Stellen Sie sicher, dass Sie jedem Status in Ihrem Meldermodell einen eindeutigen Namen geben. Der Name des Bundesstaates muss 1-128 Zeichen lang sein. Gültige Zeichen: a-z, A-Z, 0-9, _ (Unterstrich) und - (Bindestrich).

- Meldung: Die Definitionen entsprachen initialStateName *`initial-state-name`* keinem definierten Bundesstaat.

  Lösung: Möglicherweise erhalten Sie diese Fehlermeldung, wenn der ursprüngliche Statusname falsch ist. Das Detektormodell bleibt im Anfangszustand (Start), bis eine Eingabe eintrifft. Sobald eine Eingabe eintrifft, geht das Detektormodell sofort in den nächsten Zustand über. Stellen Sie sicher, dass der ursprüngliche Statusname der Name eines definierten Zustands ist und dass Sie den richtigen Namen eingeben.

- Meldung: Die Definition des Detektormodells muss mindestens eine Eingabe in einer Bedingung verwenden.

  Lösung: Dieser Fehler wird möglicherweise angezeigt, wenn Sie in einer Bedingung keine Eingabe angegeben haben. Sie müssen mindestens eine Eingabe in mindestens einer Bedingung verwenden. Andernfalls werden eingehende Daten AWS IoT Events nicht ausgewertet.

- Meldung: Es können nur Sekunden und DurationExpression eingegeben werden. SetTimer

  Lösung: Möglicherweise erhalten Sie diese Fehlermeldung, wenn Sie sowohl als auch `seconds` `durationExpression` für Ihren Timer verwendet haben. Stellen Sie sicher, dass Sie entweder `seconds` oder `durationExpression` als Parameter von verwenden`SetTimerAction`. Weitere Informationen finden Sie unter SetTimerAction in der AWS IoT Events -API-Referenz.

- Meldung: Eine Aktion in Ihrem Meldermodell ist nicht erreichbar. Überprüfen Sie den Zustand, der die Aktion auslöst.

  Lösung: Wenn eine Aktion in Ihrem Meldermodell nicht erreichbar ist, wird der Zustand des Ereignisses als falsch bewertet. Überprüfen Sie die Bedingung des Ereignisses, das die Aktion enthält, um sicherzustellen, dass sie als wahr ausgewertet wird. Wenn die Bedingung des Ereignisses als wahr ausgewertet wird, sollte die Aktion erreichbar sein.

- Meldung: Ein Eingabeattribut wird gelesen, dies kann jedoch durch den Ablauf eines Timers verursacht werden.

  Lösung: Der Wert eines Eingabeattributs kann gelesen werden, wenn einer der folgenden Fälle eintritt:

  - Ein neuer Eingabewert wurde empfangen.
  - Wenn ein Timer im Melder abgelaufen ist.

Um sicherzustellen, dass ein Eingabeattribut nur ausgewertet wird, wenn der neue Wert für diese Eingabe empfangen wird, fügen Sie einen `triggerType(“Message”)` Funktionsaufruf wie folgt in Ihre Bedingung ein:

Der ursprüngliche Zustand wird im Detektormodell ausgewertet:

```
if ($input.HeartBeat.status == “OFFLINE”)
```

würde dem Folgenden ähnlich werden:

```
if ( triggerType("MESSAGE") &&  $input.HeartBeat.status == “OFFLINE”)
```

wobei ein Aufruf der `triggerType(“Message”)` Funktion vor der in der Bedingung angegebenen ersten Eingabe erfolgt. Bei Verwendung dieser Technik ergibt die `triggerType("Message")` Funktion den Wert true und erfüllt die Bedingung, dass ein neuer Eingabewert empfangen wird. Weitere Informationen zur Verwendung der `triggerType` Funktion finden Sie `triggerType` im Abschnitt Ausdrücke im AWS IoT Events Entwicklerhandbuch

- Meldung: Ein Status in Ihrem Detektormodell ist nicht erreichbar. Überprüfen Sie den Zustand, der einen Übergang in den gewünschten Zustand bewirkt.

  Lösung: Wenn ein Zustand in Ihrem Detektormodell nicht erreichbar ist, wird eine Bedingung, die zu einem eingehenden Übergang in diesen Zustand führt, als falsch bewertet. Vergewissern Sie sich, dass die Bedingungen für die eingehenden Übergänge in diesen unerreichbaren Zustand in Ihrem Detektormodell als wahr gewertet werden, sodass der gewünschte Zustand erreichbar werden kann.

- Nachricht: Ein ablaufender Timer kann dazu führen, dass eine unerwartete Anzahl von Nachrichten gesendet wird.

  Lösung: Um zu verhindern, dass Ihr Meldermodell in einen unendlichen Zustand übergeht und eine unerwartete Anzahl von Nachrichten sendet, weil ein Timer abgelaufen ist, sollten Sie einen `triggerType("Message")` Funktionsaufruf unter den folgenden Bedingungen Ihres Meldermodells in Betracht ziehen:

  Der ursprüngliche Zustand wird im Meldermodell bewertet:

```
if (timeout("awake"))
```

würde in einen Zustand umgewandelt werden, der dem folgenden ähnelt:

```
if (triggerType("MESSAGE") && timeout("awake"))
```

wobei ein Aufruf der `triggerType(“Message”)` Funktion vor der ersten Eingabe erfolgt, die in der Bedingung bereitgestellt wird.

Diese Änderung verhindert, dass Timer-Aktionen in Ihrem Detektor ausgelöst werden, wodurch verhindert wird, dass eine Endlosschleife von Nachrichten gesendet wird. Weitere Informationen zur Verwendung von Timer-Aktionen in Ihrem Melder finden Sie auf der Seite Verwenden integrierter Aktionen im AWS IoT Events Entwicklerhandbuch

`expression-syntax`

Ein Analyseergebnis mit Informationen über`expression-syntax`, entspricht den folgenden Fehlermeldungen:

- Meldung: Ihr Payload-Ausdruck {*`expression`*} ist nicht gültig. Der definierte Payload-Typ ist JSON. Sie müssen also einen Ausdruck angeben, der AWS IoT Events eine Zeichenfolge ergibt.

  Lösung: Wenn der angegebene Payload-Typ JSON ist, wird AWS IoT Events zunächst geprüft, ob der Dienst Ihren Ausdruck als Zeichenfolge auswerten kann. Das ausgewertete Ergebnis darf weder ein Boolescher Wert noch eine Zahl sein. Wenn die Überprüfung nicht erfolgreich ist, erhalten Sie möglicherweise diesen Fehler.
- Nachricht: `SetVariableAction.value` muss ein Ausdruck sein. Der Wert " *`variable-value`* konnte nicht analysiert werden

  Lösung: Sie können es verwenden`SetVariableAction`, um eine Variable mit einem `name` und `value` zu definieren. Dabei `value` kann es sich um eine Zeichenfolge, eine Zahl oder einen booleschen Wert handeln. Sie können auch einen Ausdruck für die angeben. `value` Weitere Informationen finden Sie unter SetVariableAction, in der AWS IoT Events API-Referenz.
- Meldung: Wir konnten Ihren Ausdruck der Attribute (*`attribute-name`*) für die DynamoDB-Aktion nicht analysieren. Geben Sie den Ausdruck mit der richtigen Syntax ein.

  Lösung: Sie müssen Ausdrücke für alle Parameter in Ersatzvorlagen verwenden`DynamoDBAction`. Weitere Informationen finden Sie unter Dynamo DBAction in der AWS IoT Events API-Referenz.

- Meldung: Wir konnten Ihren Ausdruck des TableName-Ausdrucks für die DBv2 Dynamo-Aktion nicht analysieren. Geben Sie den Ausdruck mit der richtigen Syntax ein.

  Lösung: Die `tableName` Eingabe `DynamoDBv2Action` muss eine Zeichenfolge sein. Sie müssen einen Ausdruck für die verwenden`tableName`. Die Ausdrücke akzeptieren Literale, Operatoren, Funktionen, Referenzen und Substitutionsvorlagen. Weitere Informationen finden Sie unter Dynamo DBv2 Action in der AWS IoT Events API-Referenz.

- Nachricht: Wir konnten Ihren Ausdruck nicht als gültiges JSON-Format auswerten. Die DBv2 Dynamo-Aktion unterstützt nur den JSON-Nutzdatentyp.

  Lösung: Der Payload-Typ für `DynamoDBv2` muss JSON sein. Stellen Sie sicher, dass Ihr Inhaltsausdruck für die Nutzlast als gültiges JSON ausgewertet werden AWS IoT Events kann. Weitere Informationen finden Sie unter Dynamo DBv2 Action in der AWS IoT Events API-Referenz.

- Nachricht: Wir konnten Ihren Inhaltsausdruck nicht nach der Payload von analysieren. *`action-type`* Geben Sie einen Inhaltsausdruck mit der richtigen Syntax ein.

  Lösung: Der Inhaltsausdruck kann Zeichenketten ('*`string`*') und Variablen ($variable) enthalten. *`variable-name`*), Eingabewerte ($input. *`input-name`*. *`path-to-datum`*), Zeichenkettenverkettungen und Zeichenketten, die enthalten. `${}`

- Nachricht: Benutzerdefinierte Payloads dürfen nicht leer sein.

  Lösung: Möglicherweise erhalten Sie diese Fehlermeldung, wenn Sie für Ihre Aktion „Benutzerdefiniertes Payload“ ausgewählt und keinen Inhaltsausdruck in der Konsole eingegeben haben. AWS IoT Events Wenn Sie Benutzerdefinierter Payload wählen, müssen Sie unter Benutzerdefinierter Payload einen Inhaltsausdruck eingeben. Weitere Informationen finden Sie unter Payload in der AWS IoT Events API-Referenz.

- Meldung: Fehler beim Analysieren des Dauerausdrucks " *`duration-expression`* für den Timer ". *`timer-name`*

  Lösung: Das ausgewertete Ergebnis Ihres Dauerausdrucks für den Timer muss ein Wert zwischen 60 und 31622400 sein. Das ausgewertete Ergebnis der Dauer wird auf die nächste ganze Zahl abgerundet.

- Meldung: Der Ausdruck " für konnte nicht analysiert werden *`expression action-name`*

  Lösung: Möglicherweise erhalten Sie diese Meldung, wenn der Ausdruck für die angegebene Aktion eine falsche Syntax hat. Stellen Sie sicher, dass Sie einen Ausdruck mit der richtigen Syntax

eingeben. Weitere Informationen finden Sie unter Syntax zum Filtern von Gerätedaten und zum Definieren von Aktionen in AWS IoT Events.

- Nachricht: Ihr *`fieldName`* Formular `IotSitewiseAction` konnte nicht analysiert werden. Sie müssen in Ihrem Ausdruck die richtige Syntax verwenden.

  Lösung: Möglicherweise erhalten Sie diesen Fehler, wenn Sie Ihre *`fieldName`* Form nicht analysieren AWS IoT Events konnten. `IotSitewiseAction` Stellen Sie sicher, dass der einen Ausdruck *`fieldName`* verwendet, der analysiert AWS IoT Events werden kann. Weitere Informationen finden Sie unter IotSiteWiseAction in der AWS IoT Events -API-Referenz.

`data-type`

Ein Analyseergebnis mit Informationen über`data-type`, entspricht den folgenden Fehlermeldungen:

- Meldung: Der Ausdruck „Dauer“ *`duration-expression`* für den Timer *`timer-name`* ist ungültig. Er muss eine Zahl zurückgeben.

  Lösung: Möglicherweise erhalten Sie diese Fehlermeldung, wenn Sie den Dauerausdruck für Ihren Timer nicht als Zahl auswerten AWS IoT Events konnten. Stellen Sie sicher, dass Ihr Wert in eine Zahl umgewandelt werden `durationExpression` kann. Andere Datentypen, wie z. B. Boolean, werden nicht unterstützt.
- Meldung: *`condition-expression`* Der Ausdruck ist kein gültiger Bedingungsausdruck.

  Lösung: Möglicherweise erhalten Sie diese Fehlermeldung, wenn Sie Ihren Wert nicht `condition-expression` auf einen booleschen Wert auswerten AWS IoT Events konnten. Der boolesche Wert muss entweder oder sein. `TRUE` `FALSE` Stellen Sie sicher, dass Ihr Bedingungsausdruck in einen booleschen Wert konvertiert werden kann. Wenn das Ergebnis kein boolescher Wert ist, entspricht es den Aktionen, die im Ereignis angegeben wurden, `FALSE` und ruft diese nicht auf`nextState`.
- Meldung: *`reference`* Im folgenden Ausdruck wurden inkompatible Datentypen [*`inferred-types`*] gefunden: *`expression`*

  Lösung: Lösung: Alle Ausdrücke für dasselbe Eingabeattribut oder dieselbe Variable im Detektormodell müssen auf denselben Datentyp verweisen.

  Verwenden Sie die folgenden Informationen, um das Problem zu beheben:

  - Wenn Sie eine Referenz als Operanden mit einem oder mehreren Operatoren verwenden, stellen Sie sicher, dass alle Datentypen, auf die Sie verweisen, kompatibel sind.

Im folgenden Ausdruck 2 ist Integer beispielsweise ein Operand sowohl der == Operatoren als auch. && Um sicherzustellen, dass die Operanden kompatibel sind `$variable.testVariable + 1` und auf eine Ganzzahl oder Dezimalzahl verweisen `$variable.testVariable` müssen.

Außerdem 1 ist Integer ein Operand des Operators+. `$variable.testVariable`Muss daher auf eine Ganzzahl oder Dezimalzahl verweisen.

```
‘$variable.testVariable + 1 == 2 && $variable.testVariable’
```

- Wenn Sie eine Referenz als Argument verwenden, das an eine Funktion übergeben wird, stellen Sie sicher, dass die Funktion die Datentypen unterstützt, auf die Sie verweisen.

  Für die folgende `timeout("time-name")` Funktion ist beispielsweise eine Zeichenfolge mit doppelten Anführungszeichen als Argument erforderlich. Wenn Sie einen Verweis für den *`timer-name`* Wert verwenden, müssen Sie auf eine Zeichenfolge mit doppelten Anführungszeichen verweisen.

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

  > **Note**
  >
  > Wenn Sie für die `convert(type, expression)` Funktion eine Referenz für den *`type`* Wert verwenden, muss das ausgewertete Ergebnis Ihrer Referenz `String`, `Decimal`, oder sein`Boolean`.

  Weitere Informationen finden Sie unter AWS IoT Events Referenz für Eingaben und Variablen in Ausdrücken.

- Meldung: Inkompatible Datentypen [*`inferred-types`*] wurden mit verwendet*`reference`*. Dies kann zu einem Laufzeitfehler führen.

  Lösung: Möglicherweise erhalten Sie diese Warnmeldung, wenn zwei Ausdrücke für dasselbe Eingabeattribut oder dieselbe Variable auf zwei Datentypen verweisen. Stellen Sie sicher, dass Ihre Ausdrücke für dasselbe Eingabeattribut oder dieselbe Variable auf denselben Datentyp im Detektormodell verweisen.

- Meldung: Die Datentypen [*`inferred-types`*], die Sie für den Operator [*`operator`*] eingegeben haben, sind für den folgenden Ausdruck nicht kompatibel: '*`expression`*'

Lösung: Möglicherweise erhalten Sie diese Fehlermeldung, wenn Ihr Ausdruck Datentypen kombiniert, die mit einem angegebenen Operator nicht kompatibel sind. Im folgenden Ausdruck + ist der Operator beispielsweise mit den Datentypen Integer, Decimal und String kompatibel, nicht jedoch mit Operanden des Booleschen Datentyps.

```
true + false
```

Sie müssen sicherstellen, dass die Datentypen, die Sie mit einem Operator verwenden, kompatibel sind.

- Meldung: Die gefundenen Datentypen [*inferred-types*] *input-attribute* sind nicht kompatibel und können zu einem Laufzeitfehler führen.

  Lösung: Möglicherweise erhalten Sie diese Fehlermeldung, wenn zwei Ausdrücke für dasselbe Eingabeattribut auf zwei Datentypen verweisen, entweder für den `OnEnterLifecycle` eines Zustands oder für den `OnInputLifecycle` und `OnExitLifecycle` eines Zustands. Stellen Sie sicher, dass Ihre Ausdrücke in `OnEnterLifecycle` (oder, `OnInputLifecycle` sowohl als auch`OnExitLifecycle`) für jeden Status Ihres Detektormodells auf denselben Datentyp verweisen.

- Meldung: Der Nutzdatenausdruck [*expression*] ist nicht gültig. Geben Sie einen Ausdruck an, der zur Laufzeit zu einer Zeichenfolge ausgewertet würde, da der Payload-Typ das JSON-Format ist.

  Lösung: Möglicherweise erhalten Sie diesen Fehler, wenn der angegebene Payload-Typ JSON ist, aber AWS IoT Events Sie können seinen Ausdruck nicht als Zeichenfolge auswerten. Stellen Sie sicher, dass es sich bei dem ausgewerteten Ergebnis um eine Zeichenfolge handelt, nicht um einen booleschen Wert oder eine Zahl.

- Meldung: Ihr interpolierter Ausdruck {*interpolated-expression*} muss zur Laufzeit entweder eine Ganzzahl oder einen booleschen Wert ergeben. Andernfalls kann Ihr Nutzdatenausdruck {*payload-expression*} zur Laufzeit nicht als gültiges JSON analysiert werden.

  Lösung: Möglicherweise erhalten Sie diese Fehlermeldung, wenn Sie Ihren interpolierten Ausdruck nicht als Ganzzahl oder einen booleschen Wert auswerten AWS IoT Events konnten. Stellen Sie sicher, dass Ihr interpolierter Ausdruck in eine Ganzzahl oder einen booleschen Wert konvertiert werden kann, da andere Datentypen, wie z. B. Zeichenfolge, nicht unterstützt werden.

- Meldung: Der Ausdruckstyp im `IotSitewiseAction` Feld *expression* ist als Typ definiert und wird als Typ *defined-type* abgeleitet. *inferred-type* Der definierte Typ und der abgeleitete Typ müssen identisch sein.

Lösung: Möglicherweise erhalten Sie diese Fehlermeldung, wenn Ihr Ausdruck in `propertyValue` of einen Datentyp `IotSitewiseAction` hat, der anders definiert ist als der von abgeleitete Datentyp. AWS IoT Events Stellen Sie sicher, dass Sie für alle Instanzen dieses Ausdrucks in Ihrem Detektormodell denselben Datentyp verwenden.

- Meldung: Die für die `setTimer` Aktion verwendeten Datentypen [*`inferred-types`*] ergeben nicht `Integer` den folgenden Ausdruck: *`expression`*

  Lösung: Möglicherweise erhalten Sie diese Fehlermeldung, wenn der abgeleitete Datentyp für Ihren Dauerausdruck nicht Integer oder Decimal ist. Stellen Sie sicher, dass Ihr `durationExpression` Wert in eine Zahl umgewandelt werden kann. Andere Datentypen wie Boolean und String werden nicht unterstützt.

- Meldung: Die Datentypen [*`inferred-types`*], die mit Operanden des Vergleichsoperators [*`operator`*] verwendet werden, sind im folgenden Ausdruck nicht kompatibel: *`expression`*

  Lösung: Die abgeleiteten Datentypen für die Operanden von *`operator`* im bedingten Ausdruck (*`expression`*) Ihres Detektormodells stimmen nicht überein. Die Operanden müssen mit den passenden Datentypen in allen anderen Teilen Ihres Detektormodells verwendet werden.

> **Tip**
>
> Sie können sie verwendenconvert, um den Datentyp eines Ausdrucks in Ihrem Detektormodell zu ändern. Weitere Informationen finden Sie unter Funktionen zur Verwendung in Ausdrücken AWS IoT Events.

`referenced-data`

Ein Analyseergebnis mit Informationen über`referenced-data`, entspricht den folgenden Fehlermeldungen:

- Meldung: Defekter Timer erkannt: Timer *`timer-name`* wird in einem Ausdruck verwendet, aber nie gesetzt.

  Lösung: Möglicherweise erhalten Sie diese Fehlermeldung, wenn Sie einen Timer verwenden, der nicht eingestellt ist. Sie müssen einen Timer festlegen, bevor Sie ihn in einem Ausdruck verwenden können. Stellen Sie außerdem sicher, dass Sie den richtigen Timer-Namen eingeben.

- Meldung: Fehlerhafte Variable erkannt: *variable-name* Die Variable wird in einem Ausdruck verwendet, aber nie gesetzt.

  Lösung: Möglicherweise erhalten Sie diese Fehlermeldung, wenn Sie eine Variable verwenden, die nicht gesetzt ist. Sie müssen eine Variable festlegen, bevor Sie sie in einem Ausdruck verwenden können. Stellen Sie außerdem sicher, dass Sie den richtigen Variablennamen eingeben.
- Meldung: Fehlerhafte Variable erkannt: Eine Variable wird in einem Ausdruck verwendet, bevor sie auf einen Wert gesetzt wird.

  Lösung: Jeder Variablen muss ein Wert zugewiesen werden, bevor sie in einem Ausdruck ausgewertet werden kann. Legen Sie den Wert der Variablen vor jeder Verwendung fest, damit ihr Wert abgerufen werden kann. Stellen Sie außerdem sicher, dass Sie den richtigen Variablennamen eingeben.

`referenced-resource`

Ein Analyseergebnis mit Informationen über`referenced-resource`, entspricht den folgenden Fehlermeldungen:

- Meldung: Die Detector Model Definition enthält einen Verweis auf Input, der nicht existiert.

  Lösung: Möglicherweise erhalten Sie diese Fehlermeldung, wenn Sie Ausdrücke verwenden, um auf eine Eingabe zu verweisen, die nicht existiert. Stellen Sie sicher, dass Ihr Ausdruck auf eine vorhandene Eingabe verweist, und geben Sie den richtigen Eingabenamen ein. Wenn Sie keine Eingabe haben, erstellen Sie zuerst eine.
- Meldung: Die Modelldefinition des Detektors enthält folgende ungültige Angaben InputName: *input-name*

  Lösung: Möglicherweise erhalten Sie diese Fehlermeldung, wenn Ihr Detektormodell einen ungültigen Eingabenamen enthält. Stellen Sie sicher, dass Sie den richtigen Eingabenamen eingeben. Der Eingabename muss 1-128 Zeichen lang sein. Gültige Zeichen: a-z, A-Z, 0-9, _ (Unterstrich) und - (Bindestrich).

Analysieren Sie ein Detektormodell für AWS IoT Events (Konsole)

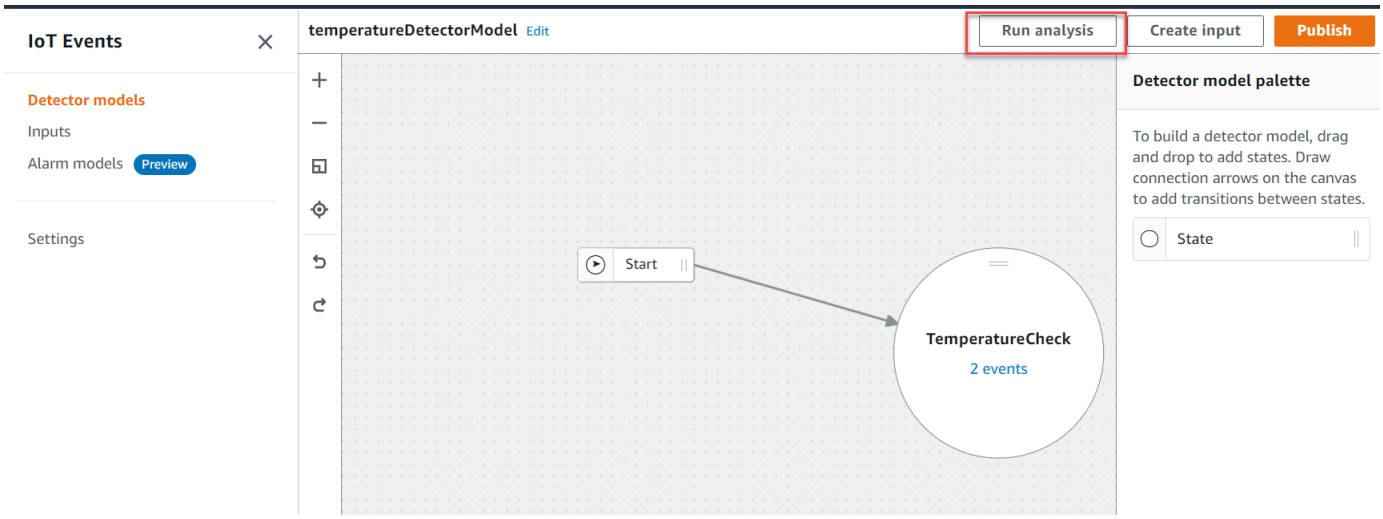
AWS IoT Events ermöglicht es Ihnen, IoT-Daten zu überwachen und darauf zu reagieren, indem Sie Ereignisse erkennen und Aktionen mit der AWS IoT Events API auslösen. In den folgenden Schritten wird die AWS IoT Events Konsole verwendet, um ein Detektormodell zu analysieren.

> **Note**
>
> Nachdem Sie AWS IoT Events mit der Analyse Ihres Detektormodells begonnen haben, haben Sie bis zu 24 Stunden Zeit, um die Analyseergebnisse abzurufen.

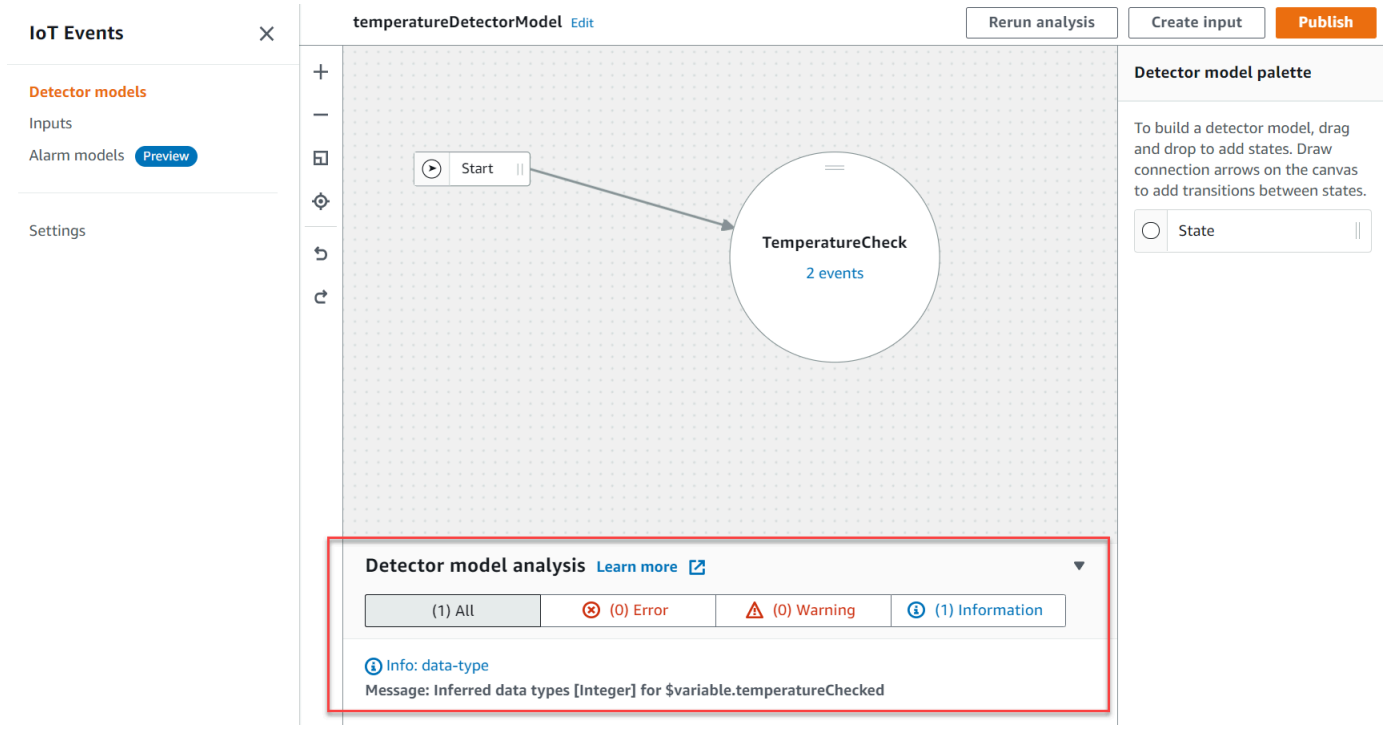
Eine Analyse des Detektormodells kann Ihnen helfen, Ihre Modelle zu optimieren, potenzielle Probleme zu identifizieren und sicherzustellen, dass sie wie vorgesehen funktionieren. In einem Windpark könnte die Analyse des Detektormodells beispielsweise aufzeigen, ob das Modell potenzielle Getriebeschäden aufgrund abnormaler Schwingungsmuster korrekt identifiziert. Oder ob das Modell präzise Wartungswarnungen auslöst, wenn die Windgeschwindigkeiten die sicheren Betriebsgrenzwerte überschreiten. Indem Sie ein auf der Analyse basierendes Modell verfeinern, können Sie die vorausschauende Wartung verbessern, Ausfallzeiten reduzieren und die Gesamteffizienz der Energieerzeugung verbessern.

Um ein Detektormodell zu analysieren

1. Melden Sie sich an der AWS IoT Events -Konsole an.
2. Wählen Sie im Navigationsbereich die Option Detektormodelle aus.
3. Wählen Sie unter Detektormodelle das Zieldetektormodell aus.
4. Wählen Sie auf der Seite mit dem Meldermodell die Option Bearbeiten aus.
5. Wählen Sie in der oberen rechten Ecke die Option Analyse ausführen aus.



Im Folgenden finden Sie ein Beispiel für ein Analyseergebnis in der AWS IoT Events Konsole.



Analysieren Sie ein Detektormodell in AWS IoT Events (AWS CLI)

Die programmatische Analyse Ihrer AWS IoT Events Detektormodelle bietet wertvolle Einblicke in deren Struktur, Verhalten und Leistung. Dieser API-basierte Ansatz ermöglicht eine automatisierte Analyse, die Integration in Ihre bestehenden Workflows und die Möglichkeit, Massenoperationen über mehrere Detektormodelle hinweg durchzuführen. Durch die Nutzung der StartDetectorModelAnalysisAPI können Sie eingehende Untersuchungen Ihrer Modelle einleiten und so potenzielle Probleme identifizieren, logische Abläufe optimieren und sicherstellen, dass Ihre IoT-Ereignisverarbeitung Ihren Geschäftsanforderungen entspricht.

In den folgenden Schritten wird AWS CLI ein Detektormodell analysiert.

Um ein Detektormodell zu analysieren mit AWS CLI

1. Führen Sie den folgenden Befehl aus, um eine Analyse zu starten.

```
aws iotevents start-detector-model-analysis --cli-input-json file://file-name.json
```

> **Note**
>
> *file-name*Ersetzen Sie es durch den Namen der Datei, die die Definition des Detektormodells enthält.

Example Definition des Detektormodells

```
{
    "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"
    }
}
```

Wenn Sie den verwenden AWS CLI , um ein vorhandenes Detektormodell zu analysieren, wählen Sie eine der folgenden Optionen, um die Definition des Detektormodells abzurufen:

- Wenn Sie die AWS IoT Events Konsole verwenden möchten, gehen Sie wie folgt vor:
  1. Wählen Sie im Navigationsbereich die Option Detector models aus.
  2. Wählen Sie unter Detektormodelle das Zieldetektormodell aus.
  3. Wählen Sie unter Aktion die Option Detektormodell exportieren aus, um das Detektormodell herunterzuladen. Das Detektormodell wird in JSON gespeichert.
  4. Öffnen Sie die JSON-Datei für das Detektormodell.
  5. Sie benötigen nur das `detectorModelDefinition` Objekt. Entfernen Sie Folgendes:
     - Die erste geschweifte Klammer ({) oben auf der Seite
     - Die Linie `detectorModel`
     - Das `detectorModelConfiguration`-Objekt
     - Die letzte geschweifte Klammer (}) unten auf der Seite
  6. Speichern Sie die Datei.
- Wenn Sie den verwenden möchten AWS CLI, gehen Sie wie folgt vor:
  1. Führen Sie folgenden Befehl von einem Terminal aus.

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

  2. *`detector-model-name`*Ersetzen Sie es durch den Namen Ihres Meldermodells.
  3. Kopieren Sie das `detectorModelDefinition` Objekt in einen Texteditor.

4. Fügen Sie geschweifte Klammern ({}) außerhalb von hinzu. `detectorModelDefinition`
5. Speichern Sie die Datei in JSON.

Example Beispielantwort

```
{
    "analysisId": "c1133390-14e3-4204-9a66-31efd92a4fed"
}
```

2. Kopieren Sie die Analyse-ID aus der Ausgabe.
3. Führen Sie den folgenden Befehl aus, um den Status der Analyse abzurufen.

```
aws iotevents describe-detector-model-analysis --analysis-id "analysis-id"
```

> **Note**
>
> *analysis-id*Ersetzen Sie durch die Analyse-ID, die Sie kopiert haben.

Example Beispielantwort

```
{
    "status": "COMPLETE"
}
```

Der Status kann einer der folgenden Werte sein:

- `RUNNING`— AWS IoT Events analysiert Ihr Detektormodell. Dieser Vorgang kann bis zu einer Minute dauern.
- `COMPLETE`— die Analyse Ihres Detektormodells AWS IoT Events abgeschlossen.
- `FAILED`— AWS IoT Events konnte Ihr Detektormodell nicht analysieren. Bitte versuchen Sie es später erneut.

4. Führen Sie den folgenden Befehl aus, um ein oder mehrere Analyseergebnisse des Detektormodells abzurufen.

> **Note**
>
> *analysis-id*Ersetzen Sie es durch die Analyse-ID, die Sie kopiert haben.

```
aws iotevents get-detector-model-analysis-results --analysis-id "analysis-id"
```

Example Beispielantwort

```
{
    "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**
>
> Nachdem Sie AWS IoT Events mit der Analyse Ihres Detektormodells begonnen haben, haben Sie bis zu 24 Stunden Zeit, um die Analyseergebnisse abzurufen.

AWS IoT Events Befehle

Dieses Kapitel enthält eine umfassende Anleitung zu allen API-Vorgängen, die in verfügbar sind AWS IoT Events. Es enthält ausführliche Erläuterungen, einschließlich Beispielanfragen, Antworten und möglicher Fehler für jeden Vorgang in den unterstützten Webdienstprotokollen. Wenn Sie diese API-Operationen verstehen, können Sie sie effektiv AWS IoT Events in Ihre IoT-Anwendungen integrieren und Ihre Workflows zur Erkennung und Reaktion auf Ereignisse automatisieren.

AWS IoT Events Aktionen

Sie können AWS IoT Events API-Befehle verwenden, um Eingaben und Detektormodelle zu erstellen, zu lesen, zu aktualisieren und zu löschen und ihre Versionen aufzulisten. Weitere Informationen finden Sie AWS IoT Events in der AWS IoT Events API-Referenz unter Aktionen und Datentypen, die von unterstützt werden.

Die AWS IoT Events Abschnitte in der AWS CLI Befehlsreferenz enthalten die AWS CLI Befehle, die Sie zur Verwaltung und Bearbeitung AWS IoT Events verwenden können.

AWS IoT Events Daten

Sie können die AWS IoT Events Daten-API-Befehle verwenden, um Eingaben an Melder zu senden, Melder aufzulisten und den Status eines Melders anzuzeigen oder zu aktualisieren. Weitere Informationen finden Sie in der AWS IoT Events API-Referenz unter den Aktionen und Datentypen, die von AWS IoT Events Data unterstützt werden.

Die AWS IoT Events Datenabschnitte in der AWS CLI Befehlsreferenz enthalten die AWS CLI Befehle, mit denen Sie AWS IoT Events Daten verarbeiten können.

Dokumenthistorie für AWS IoT Events

In der folgenden Tabelle werden die wichtigen Änderungen am AWS IoT Events Entwicklerhandbuch nach dem 17. September 2020 beschrieben. Um mehr Informationen über Aktualisierungen dieser Dokumentation zu erhalten, können Sie einen RSS-Feed abonnieren.

Änderung	Beschreibung	Datum
Hinweis zum Ende des Supports	Hinweis zum Ende des Supports: Am 20. Mai 2026 AWS wird der Support für eingestellt. AWS IoT Events Nach dem 20. Mai 2026 können Sie nicht mehr auf die AWS IoT Events Konsole oder die Ressourcen zugreifen. AWS IoT Events	20. Mai 2025
Start der Region	AWS IoT Events ist jetzt in der Region Asien-Pazifik (Mumbai) verfügbar.	30. September 2021
Start der Region	AWS IoT Events ist jetzt in der Region AWS GovCloud (USA West) verfügbar.	22. September 2021
Führen Sie Analysen durch Ausführen von Analysen zur Fehlerbehebung bei einem Detektormodell durch	AWS IoT Events kann jetzt Ihr Detektormodell analysier en und Analyseergebnisse generieren, die Sie zur Fehlerbehebung bei Ihrem Detektormodell verwenden können.	23. Februar 2021
Start in der Region	AWS IoT Events In China (Peking) eingeführt.	30. September 2020

Verwendung von Ausdrücken	Es wurden Beispiele hinzugefügt, die Ihnen zeigen, wie man Ausdrücke schreibt.	22. September 2020
Überwachung mit Alarmen	Alarme helfen Ihnen dabei, Ihre Daten auf Änderungen zu überwachen. Sie können Alarme erstellen, die Benachrichtigungen senden, wenn ein Schwellenwert überschritten wird.	1. Juni 2020

Frühere Aktualisierungen

In der folgenden Tabelle werden wichtige Änderungen am AWS IoT Events Entwicklerhandbuch vor dem 18. September 2020 beschrieben.

Änderung	Beschreibung	Datum
Die Typvalidierung wurde zur Expressionsreferenz hinzugefügt	Informationen zur Typvalidierung wurden der Ausdrucksreferenz hinzugefügt.	3. August 2020
Regionswarnung für andere Dienste hinzugefügt	Es wurde eine Warnung zur Auswahl derselben Region für AWS IoT Events und andere AWS Dienste hinzugefügt.	7. Mai 2020
Ergänzungen, Aktualisierungen	• Funktion zur Anpassung der Nutzlast • Neue Ereignisaktionen: Amazon DynamoDB und AWS IoT SiteWise	27. April 2020
Integrierte Funktionen für bedingte Ausdrücke im	Integrierte Funktionen für bedingte Ausdrücke im	10. September 2019

Änderung	Beschreibung	Datum
Detektormodell wurden hinzugefügt	Detektormodell wurden hinzugefügt.	
Beispiele für Detektormodelle hinzugefügt	Beispiele für das Detektorm odell wurden hinzugefügt.	5. August 2019
Neue Event-Aktionen hinzugefügt	Neue Event-Aktionen hinzugefügt für: • Lambda • Amazon SQS • Kinesis Data Firehose • AWS IoT Events Eingabe	19. Juli 2019
Ergänzungen, Korrekturen	• Die Beschreibung der `timeout()` Funktion wurde aktualisiert. • Bewährte Methode in Bezug auf Kontoinaktivität hinzugefügt.	11. Juni 2019
Die Berechtigungsrichtlinie und die Debug-Optionen für die Konsole wurden aktualisi ert	• Die Richtlinie für Konsolenb erechtigungen wurde aktualisiert. • Das Bild der Seite mit den Debug-Optionen für die Konsole wurde aktualisiert.	5. Juni 2019
Aktualisierungen	AWS IoT Events Der Dienst ist jetzt allgemein verfügbar.	30. Mai 2019

Änderung	Beschreibung	Datum
Ergänzungen, Aktualisierungen	• Aktualisierte Sicherheitsinformationen. • Ein Beispiel für ein kommentiertes Detektormodell wurde hinzugefügt.	22. Mai 2019
Beispiele und erforderliche Berechtigungen wurden hinzugefügt	Beispiele für Amazon SNS SNS-Nutzlasten hinzugefügt; Ergänzungen zu den erforderlichen Berechtigungen für. `CreateDetectorModel`	17. Mai 2019
Zusätzliche Sicherheitsinformationen hinzugefügt	Dem Sicherheitsbereich wurden Informationen hinzugefügt.	9. Mai 2019
Limitierte Vorschauversion	Eingeschränkte Vorschauversion der Dokumentation.	28. März 2019