aws

开发人员指南

AWS IoT Events

AWS IoT Events: 开发人员指南

Table of Contents

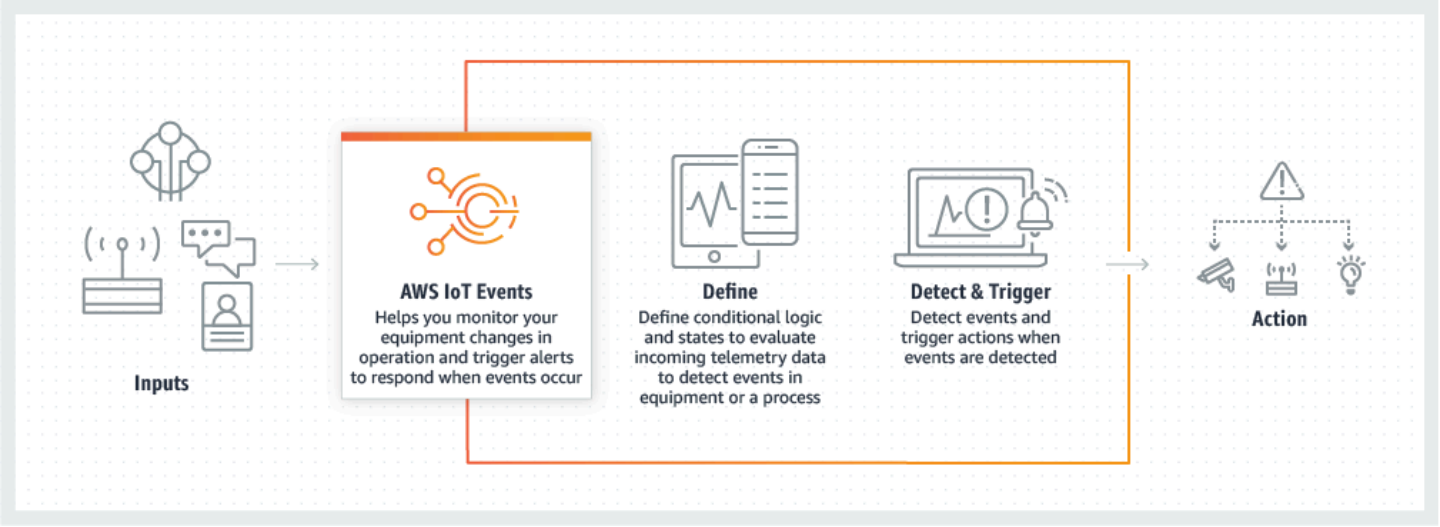
终止支持通知：2026 年 5 月 20 日， AWS 将终止对的支持。 AWS IoT Events 2026 年 5 月 20 日之后，您将无法再访问 AWS IoT Events 控制台或 AWS IoT Events 资源。有关更多信息，请参阅AWS IoT Events 终止支持。

本文属于机器翻译版本。若本译文内容与英语原文存在差异，则一律以英文原文为准。

什么是 AWS IoT Events？

AWS IoT Events 使您能够监控设备或设备群是否出现故障或运行变化，并在此类事件发生时触发操作。 AWS IoT Events 持续监视来自设备、流程、应用程序和其他 AWS 服务的物联网传感器数据，以识别重大事件，以便您采取行动。

AWS IoT Events 用于在 AWS 云中构建复杂的事件监控应用程序，您可以通过 AWS IoT Events 控制台或访问这些应用程序 APIs。



主题

优点和特点

接受来自多个来源的输入

AWS IoT Events 接受来自许多 IoT 遥测数据源的输入。其中包括传感器设备、管理应用程序和其他 AWS IoT 服务，例如 AWS IoT Core 和 AWS IoT Analytics。您可以使用标准 API 接口 (`BatchPutMessage`API) 或 AWS IoT Events 控制台将 AWS IoT Events 任何遥测数据输入推送到。

有关入门的更多信息 AWS IoT Events，请参阅AWS IoT Events 控制台入门。

使用简单的逻辑表达式识别复杂的事件模式

AWS IoT Events 可以识别涉及来自单个 IoT 设备或应用程序，或者来自不同设备和许多独立传感器的多个输入的事件模式。这特别有用，因为每个传感器和应用程序都提供重要信息。但是，只有将不同的传感器和应用程序数据结合起来，才能全面了解操作的性能和质量。您可以配置 AWS IoT Events 检测器，使其使用简单的逻辑表达式而不是复杂的代码来识别这些事件。

有关逻辑表达式的更多信息，请参阅用于筛选、转换和处理事件数据的表达式。

根据事件触发操作

AWS IoT Events 允许您在亚马逊简单通知服务 (Amazon SNS) Simple Notification、Lambda AWS IoT Core、亚马逊 SQS 和亚马逊 Kinesis Firehose 中直接触发操作。您还可以使用 AWS IoT 规则引擎触发 AWS Lambda 函数，从而可以使用其他服务（例如 Amazon Connect）或您自己的企业资源规划 (ERP) 应用程序来执行操作。

AWS IoT Events 包括您可以执行的预建操作库，还允许您定义自己的动作。

要了解有关根据事件触发操作的更多信息，请参阅支持在中接收数据和触发操作的操作 AWS IoT Events。

自动扩展以满足车队的需求

AWS IoT Events 连接同类设备时会自动缩放。您可以为特定类型的设备定义一次探测器，该服务将自动扩展和管理连接到该设备的所有实例 AWS IoT Events。

要浏览探测器模型的示例，请参阅AWS IoT Events 探测器模型示例。

使用案例

AWS IoT Events 有很多用途。以下是一些用例示例。

监控和维护远程设备

监控一组远程部署的机器可能具有挑战性，尤其是在没有明确背景的情况下发生故障时。如果一台机器停止运行，则可能意味着更换整个处理单元或机器。但这是不可持续的。借助此功能， AWS IoT Events 您可以从每台计算机上的多个传感器接收消息，以帮助您诊断一段时间内的特定问题。现在，您无需更换整个设备，而是掌握了必要的信息，可以向技术人员发送需要更换的确切部件。拥有数百万台机器，节省的费用可高达数百万美元，从而降低您拥有或维护每台机器的总成本。

管理工业机器人

在设施中部署机器人以实现包裹移动的自动化，可以大大提高效率。为了最大限度地降低成本，机器人可以配备简单、低成本的传感器，将数据报告到云端。但是，由于有数十个传感器和数百种操作模式，实时检测问题可能具有挑战性。使用 AWS IoT Events，您可以构建一个专家系统，在云端处理这些传感器数据，创建警报，以便在故障即将来临时自动通知技术人员。

跟踪楼宇自动化系统

在数据中心，监控高温和低湿度有助于防止设备故障。传感器通常从许多制造商处购买，每种类型都有自己的管理软件。但是，来自不同供应商的管理软件有时不兼容，因此很难发现问题。通过使用 AWS IoT Events，您可以设置警报，以便在出现故障之前提前将供暖和冷却系统出现问题通知运营分析师。通过这种方式，您可以防止数据中心计划外停机，从而避免数千美元的设备更换费用和潜在的收入损失。

AWS IoT Events 支持终止

经过深思熟虑，我们决定终止对该 AWS IoT Events 服务的支持，自 2026 年 5 月 20 日起生效。 AWS IoT Events 从 2025 年 5 月 20 日起，将不再接受新客户。作为拥有账户在 2025 年 5 月 20 日之前注册该服务的现有客户，您可以继续使用这些 AWS IoT Events 功能。2026 年 5 月 20 日之后，您将无法再使用 AWS IoT Events。

本页为 AWS IoT Events 客户提供了过渡到替代解决方案以满足您的业务需求的说明和注意事项。

> **Note**
>
> 这些指南中介绍的解决方案旨在作为说明性示例，而不是用于生产的功能替代品。 AWS IoT Events 根据您的业务需求自定义代码、工作流程和相关 AWS 资源。

主题

迁出时的注意事项 AWS IoT Events

- 实施安全最佳实践，包括使用对每个组件具有最低权限的 IAM 角色以及对静态和传输中的数据进行加密。有关更多信息，请参阅《IAM 用户指南》中的 IAM 安全最佳实践。
- 根据您的数据摄取要求考虑 Kinesis 流的分片数量。有关 Kinesis 分片的更多信息，请参阅亚马逊 Kinesis Data Streams 开发者指南中的亚马逊 Kinesis Dat a Stre ams 术语和概念。
- 使用指标和日志设置全面 CloudWatch 的监控和调试。有关更多信息，请参阅什么是 CloudWatch？在《亚马逊 CloudWatch 用户指南》中。
- 考虑一下错误处理的结构，包括如何管理重复处理失败的消息、实施重试策略以及如何设置隔离和分析有问题的消息的流程。
- 使用定AWS 价计算器估算特定用例的成本。

中探测器模型的迁移程序 AWS IoT Events

本节介绍替代解决方案，这些解决方案可在您迁移到其他探测器模型时提供类似的探测器模型功能 AWS IoT Events。

您可以通过 AWS IoT Core 规则将数据摄取迁移到其他 AWS 服务的组合。可以将数据路由到 MQT BatchPutMessageT 主题，而不是通过 API 获取数据。 AWS IoT Core

这种迁移方法利用 AWS IoT Core MQTT 主题作为物联网数据的入口点，取代了直接输入。 AWS IoT Events选择 MQTT 主题有几个关键原因。由于 MQTT 在行业中的广泛使用，它们与物联网设备具有广泛的兼容性。这些主题可以处理来自众多设备的大量消息，从而确保可扩展性。它们还允许根据内容或设备类型灵活地路由和筛选消息。此外， AWS IoT Core MQTT 主题与其他 AWS 服务无缝集成，从而简化了迁移过程。

数据从 MQTT 主题流入一个架构，该架构结合了 Amazon Kinesis Data Streams、 AWS Lambda 一个函数、一个亚马逊 DynamoDB 表和亚马逊计划。 EventBridge 这种服务组合复制并增强了以前提供的功能 AWS IoT Events，使您可以更加灵活地控制物联网数据处理管道。

比较架构

当前 AWS IoT Events 架构通过 AWS IoT Core 规则和 `BatchPutMessage` API 提取数据。该架构 AWS IoT Core 用于数据摄取和事件发布，消息通过 AWS IoT Events 输入路由到定义状态逻辑的探测器模型。IAM 角色管理必要的权限。

新的解决方案 AWS IoT Core 用于数据摄取（现在有专门的输入和输出 MQTT 主题）。它引入了用于数据分区的 Kinesis Data Streams 和用于状态逻辑的评估器 Lambda 函数。设备状态现在存储在 DynamoDB 表中，增强的 IAM 角色管理这些服务的权限。

用途	解决方案	差异
数据摄取 — 从物联网设备接收数据	AWS IoT Core	现在需要两个不同的 MQTT 主题：一个用于摄取设备数据，另一个用于发布输出事件
消息方向-将传入的消息路由到相应的服务	AWS IoT Core 邮件路由规则	保持相同的路由功能，但现在将消息定向到 Kinesis Data Streams 而不是 AWS IoT Events
数据处理-处理和组织传入的数据流	Kinesis Data Streams	取代 AWS IoT Events 输入功能，使用设备 ID 分区提供数据摄取，用于消息处理

用途	解决方案	差异
逻辑评估-处理状态变化并触发操作	评估者 Lambda	取代 AWS IoT Events 探测器模型，通过代码而不是可视化工作流程提供可自定义的状态逻辑评估
状态管理-维护设备状态	DynamoDB 表	提供设备状态永久存储的新组件，取代了内部 AWS IoT Events 状态管理
安全-管理服务权限	IAM 角色	更新后的权限现在包括访问 Kinesis Data Streams、DynamoDB 以及 EventBridge 现有权限 AWS IoT Core

步骤 1：（可选）导出 AWS IoT Events 探测器型号配置

在创建新资源之前，请导出您的 AWS IoT Events 探测器模型定义。它们包含您的事件处理逻辑，可以作为实施新解决方案的历史参考。

Console

使用执行以下步骤以导出您的探测器模型配置： AWS IoT Events AWS 管理控制台

要使用导出探测器模型 AWS 管理控制台

1. 登录 AWS IoT Events 控制台。
2. 在左侧导航窗格中，选择探测器模型。
3. 选择要导出的探测器型号。
4. 选择导出。阅读有关输出的信息消息，然后再次选择 “导出”。
5. 对要导出的每个探测器模型重复该过程。

包含探测器模型的 JSON 输出的文件已添加到浏览器的下载文件夹中。您可以选择保存每个探测器模型配置以保留历史数据。

AWS CLI

使用运行以下命令导出您的探测器模型配置： AWS CLI

要使用导出探测器模型 AWS CLI

1. 列出您账户中的所有探测器型号：

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

2. 对于每个探测器型号，通过运行以下命令导出其配置：

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

3. 保存每个探测器型号的输出。

步骤 2：创建 IAM 角色

创建 IAM 角色以提供复制功能的权限 AWS IoT Events。本示例中的角色授予访问 DynamoDB 以进行状态管理 EventBridge、调度、Kinesis Data Streams 用于数据 AWS IoT Core 摄取、发布消息和记录的权限。 CloudWatch 这些服务共同起到替代作用 AWS IoT Events。

1. 创建具有以下权限的 IAM 角色。有关创建 IAM 角色的更多详细说明，请参阅 I A M 用户指南中的创建角色以向 AWS 服务委派权限。

   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. 添加以下 IAM 角色信任策略。信任策略允许指定的 AWS 服务担任 IAM 角色，以便它们可以执行必要的操作。有关创建 IAM 信任策略的更多详细说明，请参阅 I AM 用户指南中的使用自定义信任策略创建角色。

   JSON

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

第 3 步：创建 Amazon Kinesis Data Streams

使用或创建 Amazon Kinesis Data Streams AWS 管理控制台 。 AWS CLI

Console

要使用创建 Kinesis 数据流 AWS 管理控制台，请按照《Amazon Kinesis Data Streams 开发者指南》中创建数据流页面上的步骤进行操作。

根据您的设备数量和消息负载大小调整分片计数。

AWS CLI

使用 AWS CLI创建 Amazon Kinesis Data Streams，从您的设备中提取和分区数据。

在本次迁移中使用 Kinesis Data Streams 来取代的数据提取功能。 AWS IoT Events它提供了一种可扩展且高效的方法来收集、处理和分析来自物联网设备的实时流数据，同时还提供灵活的数据处理以及与其他 AWS 服务的集成。

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

根据您的设备数量和消息负载大小调整分片计数。

步骤 4：创建或更新 MQTT 消息路由规则

您可以创建新的 MQTT 消息路由规则或更新现有规则。

Console

1. 确定是否需要新的 MQTT 消息路由规则，或者是否可以更新现有规则。
2. 打开 AWS IoT Core 控制台。
3. 在导航窗格中，选择 “消息路由”，然后选择 “规则”。
4. 在管理部分，选择邮件路由，然后选择规则。
5. 选择 Create rule（创建规则）。
6. 在指定规则属性页面上，输入 AWS IoT Core 规则名称的规则名称。在 “规则描述-可选” 中，输入描述以标识您正在处理事件并将其转发到 Kinesis Data Stre ams。
7. 在 “配置 SQL 语句” 页上，为 SQL 语句输入以下内容：**`SELECT * FROM 'your-database'`**，然后选择下一步。
8. 在附加规则操作页面和规则操作下，选择 k inesis。
9. 为直播选择你的 Kinesis 直播。对于分区键，输入 **`your-instance-id`**。为 IAM 角色选择相应的角色，然后选择添加规则操作。

有关更多信息，请参阅创建 AWS IoT 规则以将设备数据路由到其他服务。

AWS CLI

1. 使用以下内容创建 JSON 文件 。此 JSON 配置文件定义了一 AWS IoT Core 条规则，该规则使用实例 ID 作为分区键，从主题中选择所有消息并将其转发到指定的 Kinesis 流。

```
{
```

```
    "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. 使用创建 MQTT 主题规则。 AWS CLI此步骤使用使用`events_rule.json`文件中定义的配置创建 AWS IoT Core 主题规则。 AWS CLI

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

步骤 5：获取目标 MQTT 主题的终端节点

使用目标 MQTT 主题配置您的主题发布传出消息的位置，取代之前由 AWS IoT Events处理的功能。终端节点是您的 AWS 账户和地区所独有的。

Console

1. 打开 AWS IoT Core 控制台。
2. 在左侧导航面板的 Connect 部分，选择域配置。
3. 选择 IOT: data-ATS 域配置以打开配置的详细信息页面。
4. 复制域名值。此值是终端节点。保存终端节点值，因为您将在以后的步骤中使用它。

AWS CLI

运行以下命令以获取用于发布账户传出消息的 AWS IoT Core 终端节点。

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

第 6 步：创建亚马逊 DynamoDB 表

Amazon DynamoDB 表取代了 AWS IoT Events的状态管理功能，为在新解决方案架构中保留和管理设备状态和探测器模型逻辑提供了一种可扩展且灵活的方式。

Console

创建 Amazon DynamoDB 表以保留探测器模型的状态。有关更多信息，请参阅亚马逊 DynamoDB 开发者指南中的在 DynamoDB 中创建表。

使用以下内容获取表格的详细信息：

- 在表名中，输入您选择的表名。
- 对于分区密钥，请输入您自己的实例 ID。
- 您可以使用 “表格” 设置的 “默认” 设置

AWS CLI

运行以下命令创建 DynamoDB 表。

```
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 \
```

步骤 7：创建 AWS Lambda 函数（控制台）

Lambda 函数用作核心处理引擎，取代了的探测器模型评估逻辑。 AWS IoT Events在示例中，我们与其他 AWS 服务集成，以根据您定义的规则处理传入数据、管理状态和触发操作。

使用运行时创建一个 Lambda 函数。NodeJS使用以下代码片段，替换硬编码的常量：

1. 打开 AWS Lambda console。
2. 选择创建函数。

3. 输入函数名称的名称。
4. 选择 NodeJS 22.x 作为运行时间。
5. 在更改默认执行角色下拉列表中，选择使用现有角色，然后选择您在之前的步骤中创建的 IAM 角色。
6. 选择创建函数。
7. 替换硬编码常量后，粘贴以下代码片段。
8. 函数创建后，在 “代码” 选项卡下，粘贴以下代码示例，将**your-destination-endpoint**端点替换为自己的端点。

```
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
        ]
    );
```

第 8 步：添加 Amazon Kinesis Data Streams 触发器

使用或将 Kinesis Data Streams 触发器添加到 Lambda 函数中。 AWS 管理控制台 AWS CLI

在您的 Lambda 函数中添加 Kinesis Data Streams 触发器可在您的数据摄取管道和处理逻辑之间建立连接，使其能够自动评估传入的物联网数据流并实时对事件做出反应，类似于处理输入的方式。 AWS IoT Events

Console

有关更多信息，请参阅AWS Lambda 开发人员指南中的创建事件源映射以调用 Lambda 函数。

使用以下内容了解事件源映射的详细信息：

- 在函数名称中，输入中使用的 lambda 名称。步骤 7：创建 AWS Lambda 函数（控制台）
- 对于消费者-可选，请输入您的 Kinesis 直播的 ARN。
- 对于批处理大小，输入 **10**。

AWS CLI

运行以下命令创建 Lambda 函数触发器。

```
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
```

步骤 9：测试数据摄取和输出功能 ()AWS CLI

根据您在探测器模型中定义的内容向 MQTT 主题发布有效负载。以下是 MQTT 主题的示例负载`your-topic-name`，用于测试实现。

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

您应该会看到一条发布到主题的 MQTT 消息，其中包含以下（或类似）内容：

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

```
}
```

中 AWS IoT SiteWise 警报的迁移程序 AWS IoT Events

本节介绍替代解决方案，这些解决方案在您迁移后可提供类似的警报功能 AWS IoT Events。

对于使用 AWS IoT Events 警报的 AWS IoT SiteWise 属性，您可以使用警 CloudWatch报迁移到解决方案。这种方法提供了强大的监控功能，以及诸如异常检测 SLAs 和分组警报之类的既定功能和其他功能。

比较架构

当前 AWS IoT SiteWise 属性的 AWS IoT Events 警报配置需要`AssetModelCompositeModels`在资产模型中创建，如AWS IoT SiteWise 用户指南中的定义外部警报中所述。 AWS IoT SiteWise对新解决方案的修改通常通过 AWS IoT Events 控制台进行管理。

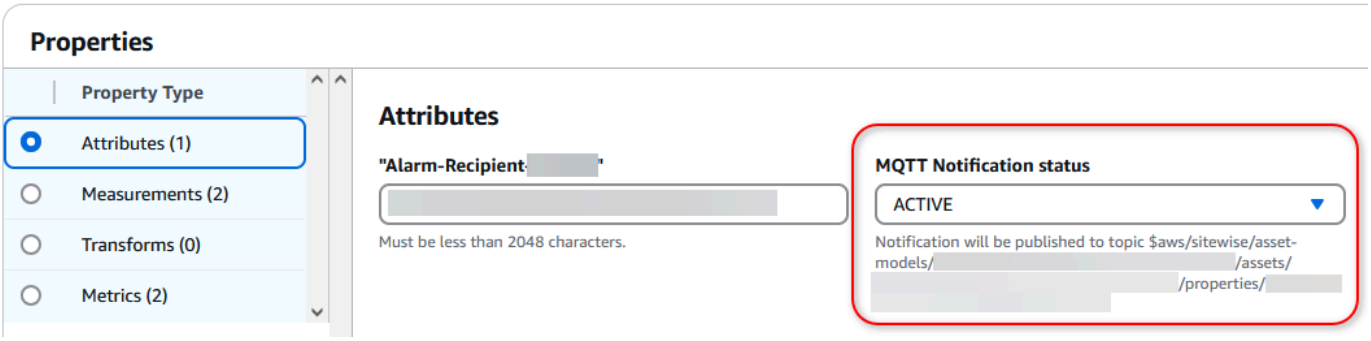
新的解决方案利用警报提供 CloudWatch 警报管理。此方法使用 AWS IoT SiteWise 通知将属性数据点发布到 AWS IoT Core MQTT 主题，然后由 Lambda 函数进行处理。该功能将这些通知转换为 CloudWatch 指标，从而通过 CloudWatch的警报框架实现警报监控。

用途	解决方案	差异
数据源-属性数据来自 AWS IoT SiteWise	AWS IoT SiteWise MQTT 通知	将直接的 IoT Events 集成替换为来自 AWS IoT SiteWise 属性的 MQTT 通知
数据处理-转换属性数据	Lambda 函数	处理 AWS IoT SiteWise 属性通知并将其转换为 CloudWatch 指标
警报评估-监控指标并触发警报	亚马逊 CloudWatch 警报	用 AWS IoT Events 警报取代 CloudWatch 警报，提供其他功能，例如异常检测
集成 — 连接 AWS IoT SiteWise	AWS IoT SiteWise 外部警报	可选功能，可将 CloudWatch 警报 AWS IoT SiteWise 作为外部警报导回去

步骤 1：在资产属性上启用 MQTT 通知

如果您使用 AWS IoT Events 集成的 AWS IoT SiteWise 警报，则可以为要监控的每个属性启用 MQTT 通知。

1. 按照中的配置资产警报 AWS IoT SiteWise程序进行操作，直到完成编辑资产模型属性的步骤。
2. 对于要迁移的每个属性，将 MQTT 通知状态更改为 “活动”。



3. 记下每个修改后的警报属性的警报发布到的主题路径。

有关更多信息，请参阅以下文档资源：

- 在《AWS IoT SiteWise 用户指南》中了解 MQTT 主题中的资产属性。
- 《AWS IoT 开发人员指南》中的 MQTT 主题。

步骤 2：创建 AWS Lambda 函数

创建一个 Lambda 函数，用于读取 MQTT 主题发布的 TQV 数组，并将单个值发布到。 CloudWatch我们将使用此 Lambda 函数作为在 AWS IoT Core 消息规则中触发的目标操作。

1. 打开 AWS Lambda console。
2. 选择创建函数。
3. 输入函数名称的名称。
4. 选择 NodeJS 22.x 作为运行时间。
5. 在更改默认执行角色下拉列表中，选择使用现有角色，然后选择您在之前的步骤中创建的 IAM 角色。

> **Note**
>
> 此过程假设您已经迁移了探测器模型。如果您没有 IAM 角色，请参阅???。

6. 选择创建函数。
7. 替换硬编码常量后，粘贴以下代码片段。

```
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)}')
        }
```

步骤 3：创建 AWS IoT Core 邮件路由规则

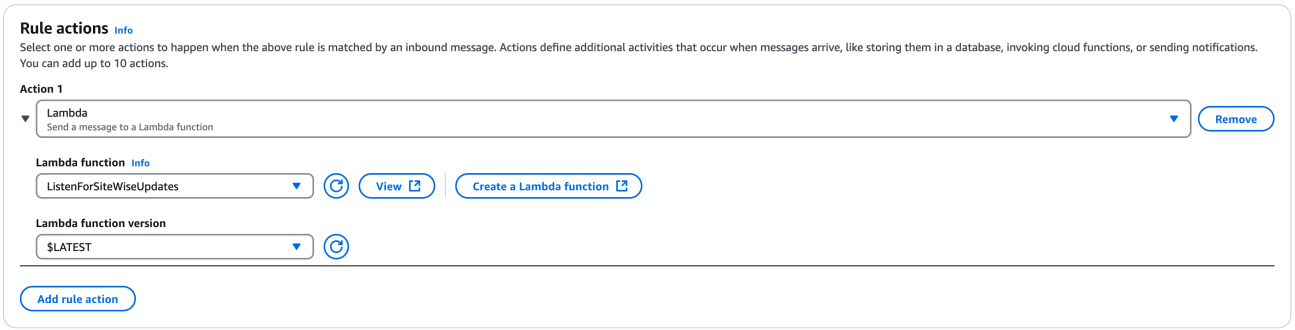
- 按照教程：重新发布 MQTT 消息过程进行操作，在出现提示时输入以下信息：

  a. 命名消息路由规则`SiteWiseToCloudwatchAlarms`。

  b. 对于查询，您可以使用以下内容：

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

  c. 在规则操作中，选择 Lambda 操作以将生成的数据发送到 AWS IoT SiteWise 。 CloudWatch 例如：



步骤 4：查看 CloudWatch 指标

当您将数据提取到之前选择的属性时 AWS IoT SiteWise，会将数据路由到我们在中步骤 1：在资产属性上启用 MQTT 通知创建的 Lambda 函数。步骤 2：创建 AWS Lambda 函数在此步骤中，您可以查看 Lambda 是否将您的指标发送到。 CloudWatch

1. 打开 CloudWatch AWS 管理控制台。
2. 在左侧导航栏中，选择指标，然后选择所有指标。
3. 选择警报的 URL 将其打开。
4. 在 S ourc e 选项卡下， CloudWatch 输出类似于此示例。此源信息确认指标数据正在输入到 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"
}
```

步骤 5：创建 CloudWatch 警报

按照 Amazon CloudWatch 用户指南中的基于静态阈值创建 CloudWatch警报程序，为每个相关指标创建警报。

> **Note**
>
> Amazon 中有许多警报配置选项。有关 CloudWatch 警报 CloudWatch 的更多信息，请参阅《亚马逊 CloudWatch 用户指南》中的 “使用亚马逊 CloudWatch 警报”。

步骤 6：（可选）将 CloudWatch 警报导入 AWS IoT SiteWise

您可以将 CloudWatch 警报配置为 AWS IoT SiteWise 使用 CloudWatch 警报操作和 Lambda 将数据发送回去。这种集成使您能够在 Monito SiteWise r 门户中查看警报状态和属性。

1. 将外部警报配置为资产模型中的一个属性。有关更多信息，请参阅《AWS IoT SiteWise 用户指南》 AWS IoT SiteWise中的定义外部警报。
2. 创建一个 Lambda 函数，该函数使用AWS IoT SiteWise 用户指南中的 BatchPutAssetPropertyValueAPI 向其发送警报数据。 AWS IoT SiteWise

3. 设置 CloudWatch 警报操作，以便在警报状态发生变化时调用 Lambda 函数。有关更多信息，请参阅《Amazon CloudWatch 用户指南》中的 “警报操作” 部分。

设置 AWS IoT Events

本节提供设置指南 AWS IoT Events，包括创建 AWS 账户、配置必要的权限以及建立用于管理资源访问权限的角色。

主题

设置一个 AWS 账户

注册获取 AWS 账户

如果您没有 AWS 账户，请完成以下步骤来创建一个。

报名参加 AWS 账户

1. 打开https://portal.aws.amazon.com/billing/注册。
2. 按照屏幕上的说明操作。

   在注册时，将接到电话或收到短信，要求使用电话键盘输入一个验证码。

   当您注册时 AWS 账户，就会创建AWS 账户根用户一个。根用户有权访问该账户中的所有 AWS 服务 和资源。作为最佳安全实践，请为用户分配管理访问权限，并且只使用根用户来执行需要根用户访问权限的任务。

AWS 注册过程完成后会向您发送一封确认电子邮件。您可以随时前往 https://aws.amazon.com/并选择“我的账户”，查看当前账户活动并管理您的账户。

创建具有管理访问权限的用户

注册后，请保护您的安全 AWS 账户 AWS 账户根用户 AWS IAM Identity Center，启用并创建管理用户，这样您就不会使用 root 用户执行日常任务。

保护你的 AWS 账户根用户

1. 选择 Root 用户并输入您的 AWS 账户 电子邮件地址，以账户所有者的身份登录。AWS 管理控制台在下一页上，输入您的密码。

要获取使用根用户登录方面的帮助，请参阅《AWS 登录 用户指南》中的 Signing in as the root user。

2. 为您的根用户启用多重身份验证（MFA）。

   有关说明，请参阅 I A M 用户指南中的为 AWS 账户 根用户启用虚拟 MFA 设备（控制台）。

创建具有管理访问权限的用户

1. 启用 IAM Identity Center。

   有关说明，请参阅《AWS IAM Identity Center 用户指南》中的启用 AWS IAM Identity Center。

2. 在 IAM Identity Center 中，为用户授予管理访问权限。

   有关使用 IAM Identity Center 目录 作为身份源的教程，请参阅《用户指南》 IAM Identity Center 目录中的使用默认设置配置AWS IAM Identity Center 用户访问权限。

以具有管理访问权限的用户身份登录

- 要使用您的 IAM Identity Center 用户身份登录，请使用您在创建 IAM Identity Center 用户时发送到您的电子邮件地址的登录 URL。

  有关使用 IAM Identity Center 用户登录的帮助，请参阅AWS 登录 用户指南中的登录 AWS 访问门户。

将访问权限分配给其他用户

1. 在 IAM Identity Center 中，创建一个权限集，该权限集遵循应用最低权限的最佳做法。

   有关说明，请参阅《AWS IAM Identity Center 用户指南》中的 Create a permission set。

2. 将用户分配到一个组，然后为该组分配单点登录访问权限。

   有关说明，请参阅《AWS IAM Identity Center 用户指南》中的 Add groups。

为设置权限 AWS IoT Events

实现适当的权限对于安全有效地使用非常重要 AWS IoT Events。本节介绍使用的某些功能所需的权限 AWS IoT Events。您可以使用 AWS CLI 命令或 AWS Identity and Access Management (IAM) 控制台创建角色和相关的权限策略，以访问资源或在中执行某些功能 AWS IoT Events。

I AM 用户指南提供了有关安全控制 AWS 资源访问权限的更多详细信息。有关特定信息 AWS IoT Events，请参阅的操作、资源和条件键 AWS IoT Events。

要使用 IAM 控制台创建和管理角色和权限，请参阅 IAM 教程：使用 IAM 角色跨 AWS 账户委派访问权限。

> **Note**
>
> 密钥可以是 1-128 个字符，可包括：
>
> - 大写字母或小写字母 a-z
> - 数字 (0-9)
> - 特殊字符-、_ 或 :。

的操作权限 AWS IoT Events

AWS IoT Events 使您能够触发使用其他 AWS 服务的操作。为此，您必须授予代表您执行这些操作的 AWS IoT Events 权限。本节包含操作列表和示例策略，该策略授权对您的资源执行所有此类操作。根据需要更改*region*和*account-id*引用。如有可能，您还应该更改通配符 (*)，以引用特定的待访问资源。您可以使用 IAM 控制台授 AWS IoT Events 予发送您定义的 Amazon SNS 提醒的权限。

AWS IoT Events 支持以下允许您使用计时器或设置变量的操作：

- `setTimer` 创建计时器。
- `resetTimer` 重置计时器。
- `clearTimer` 删除计时器。
- `setVariable` 创建变量。

AWS IoT Events 支持以下允许您使用 AWS 服务的操作：

- `iotTopicPublish` 发布有关 MATT 主题的消息。

- `iotEvents` 将数据以输入值的形式发送至 AWS IoT Events 。
- `iotSiteWise` 将数据发送至 AWS IoT SiteWise中的资产属性。
- `dynamoDB` 向 Amazon DynamoDB 表发送数据。
- `dynamoDBv2` 向 Amazon DynamoDB 表发送数据。
- `firehose`将数据发送到 Amazon Data Firehose 流。
- `lambda` 调用 AWS Lambda 函数。
- `sns` 将数据作为推送通知发送。
- `sqs` 将数据发布至 Amazon SQS 队列。

Example Policy

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:*"
        }
    ]
}
```

保护输入数据 AWS IoT Events

务必考虑谁可以授予访问供在探测器模型中使用的输入数据的权限。如果您想限制某个用户或实体的总体权限，但允许其创建或更新探测器模型，则必须授权此用户或实体更新输入路由的权限。这意味着，除为授iotevents:CreateDetectorModel和iotevents:UpdateDetectorModel授予权限外，您还必须为iotevents:UpdateInputRouting授予权限。

Example

以下是为 iotevents:UpdateInputRouting 添加权限的策略。

JSON

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

```
            "Sid": "updateRoutingPolicy",
            "Effect": "Allow",
            "Action": [
                "iotevents:UpdateInputRouting"
            ],
            "Resource": "*"
        }
    ]
}
```

您可以为 “” 指定输入的 Amazon 资源名称 (ARNs) 列表，而不是通配符 * “”，从而将此权限限制为特定输入。Resource这可以让您限制对探测器模型使用并由用户或实体创建或更新的输入数据的访问权限。

适用于 Amazon 的 CloudWatch 日志角色政策 AWS IoT Events

以下策略文件提供了允许代表您向 CloudWatch 其提交日志 AWS IoT Events 的角色策略和信任策略。

角色策略：

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

```
}
```

信任策略：

JSON

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

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

您还需要附加至 IAM 用户的 IAM 权限策略，以允许该用户按以下方式传递角色。有关更多信息，请参阅 IAM 用户指南中的授予用户向 AWS 服务传递角色的权限。

JSON

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

```
    ]
}
```

您可以使用以下命令为 CloudWatch 日志设置资源策略。这允许 AWS IoT Events 将日志事件放入 CloudWatch 流中。

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

使用以下命令放置日志记录选项。将 `roleArn` 替换为您创建的日志记录角色。

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

适用于 Amazon SNS 消息传递角色策略 AWS IoT Events

AWS IoT Events 与 Amazon SNS 集成需要仔细的权限管理，才能安全高效地发送通知。本指南将引导您完成配置 IAM 角色和策略 AWS IoT Events 以允许向 Amazon SNS 主题发布消息的过程。

以下策略文档提供了角色策略和信任策略，它们可以让 AWS IoT Events 发送 SNS 消息。

角色策略：

JSON

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

```
        }
    ]
}
```

信任策略：

JSON

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

AWS IoT Events 控制台入门

本节说明了如何使用 AWS IoT Events 控制台创建输入和探测器模型。您可以对引擎的两种状态进行建模：正常状态和超压状态。当引擎中测得的压力超过一定阈值时，模型会从正常状态过渡到超压状态。然后，它会发送 Amazon SNS 消息，提醒技术人员注意情况。当连续三个压力读数降至阈值以下时，模型将恢复到正常状态并发送另一条 Amazon SNS 消息作为确认。

我们检查连续三个低于压力阈值的读数，以消除在非线性恢复阶段或压力读数异常的情况下可能出现的超压或正常消息的卡顿现象。

在控制台上，您还可以找到几个可以自定义的预制探测器模型模板。您还可以使用控制台导入其他人编写的探测器模型或导出您的探测器模型并在不同的 AWS 区域中使用它们。如果您导入探测器模型，请确保为新区域创建所需的输入或重新创建它们，并更新所 ARNs 使用的任何角色。

使用 AWS IoT Events 控制台了解以下内容。

定义输入

要监视您的设备和流程，它们必须具有将遥测数据导入 AWS IoT Events的方法。这是通过向发送消息作为输入来完成的 AWS IoT Events。有几种方式可以实现：

- 使用 BatchPutMessage 操作。
- 在中 AWS IoT Core，为将您的消息数据转发到 AWS IoT Events的 AWS IoT 规则引擎编写一条AWS IoT Events 操作规则。您必须按名称识别输入。
- 在中 AWS IoT Analytics，使用 CreateDataset操作创建数据集`contentDeliveryRules`。这些规则指定了自动将数据集内容发送到哪个 AWS IoT Events 输入。

在您的设备以这种方式发送数据之前，您必须定义一个或多个输入。为此，请为每个输入指定一个名称，并指定输入监视传入消息数据中的哪些字段。

创建探测器模型

使用状态创建一个探测器模型（您的设备或进程的模型）。对于每种状态，请定义条件（布尔值）逻辑，该逻辑评估传入的输入以检测重要事件。当检测器模型检测到事件时，它可以使用其他 AWS 服务更改状态或启动自定义或预定义的操作。您可以定义其他事件，这些事件将在进入或退出某个状态以及满足某个条件（可选）时发起操作。

在本教程中，您将学习在模型进入或退出特定状态时，如何发送作为操作的 Amazon SNS 消息。

监视设备或进程

如果您监视多个设备或进程，请在每个输入中指定一个字段，用于标识输入来自哪个特定设备或进程。请参见`CreateDetectorModel`中的 `key` 字段。当由 `key` 标识的输入字段识别出一个新值时，就会识别出一个新设备并创建探测器。每个探测器都是一个探测器模型实例。新的探测器会继续响应来自该设备的输入，直到其探测器模型被更新或删除。

如果您监控单个进程（即使多个设备或子进程正在发送输入），也不会指定唯一的标识 `key` 字段。在这种情况下，当第一个输入到达时，模型会创建一个探测器（实例）。

将消息作为输入发送至您的探测器模型

您可通过多种方法，将来自设备或进程的消息作为输入发送至 AWS IoT Events 探测器，它不需要您对消息执行其他格式化操作。在本教程中，您将使用 AWS IoT 控制台为将消息数据转发到 AWS IoT Events的 AWS IoT 规则引擎编写AWS IoT Events 操作规则。

为此，请按名称识别输入，然后继续使用 AWS IoT 控制台生成作为输入转发到的消息 AWS IoT Events。

> **Note**
>
> 本教程使用控制台创建相同的 `input` 和 `detector model`，如 AWS IoT Events 用例教程 中的示例所示。您可以使用此 JSON 示例来帮助您学习本教程。

主题

入门必备条件 AWS IoT Events

如果您没有 AWS 帐户，请创建一个。

1. 按照中的步骤操作设置 AWS IoT Events，确保账户设置和权限正确。
2. 创建两个 Amazon Simple Notification Service (Amazon SNS) 主题。

本教程（以及相应的示例）假设您创建了两个 Amazon SNS 主题。这些主题显示为：`arn:aws:sns:us-east-1:123456789012:underPressureAction`和`arn:aws:sns:us-east-1:123456789012:pressureClearedAction`。 ARNs 将这些值替换为您创建 ARNs 的 Amazon SNS 主题。有关更多信息，请参阅《Amazon Simple Notification Service 开发人员指南》https://docs.aws.amazon.com/sns/latest/dg/。

除了向 Amazon SNS 主题发布警报之外，您还可以让探测器发送带有您指定主题的 MQTT 消息。使用此选项，您可以使用 C AWS IoT ore 控制台订阅和监控发送到这些 MQTT 主题的消息，从而验证您的检测器模型是否正在创建实例，以及这些实例是否在发送警报。您还可以使用在探测器模型中创建的输入或变量在运行时系统动态地定义 MQTT 主题名称。

3. 选择支持 AWS 区域 的 AWS IoT Events。有关更多信息，请参阅AWS 一般参考 中的 AWS IoT Events。如需帮助，请参阅《入门指南》 AWS 管理控制台中的服务入门 AWS 管理控制台。

在中为模型创建输入 AWS IoT Events

在为模型构建输入时，我们建议您收集包含示例消息负载的文件，您的设备或进程发送这些文件以报告其运行状况。拥有这些文件可以帮助您定义所需的输入。

您可以通过本节中描述的多种方法来创建输入。

创建一个 JSON 输入文件

1. 首先，请在本地文件系统上创建一个名为 `input.json` 的文件，其中包含以下内容：

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

2. 现在，您有了这个入门`input.json`文件，可以创建一个输入。有两种方法可以创建输入。您可以使用AWS IoT Events 控制台中的导航窗格创建输入。或者，也可以在探测器模型创建后在其中创建输入。

创建和配置输入

学习如何为警报模型或探测器模型创建输入。

1. 登录AWS IoT Events 控制台或选择 “创建新 AWS IoT Events 帐户” 选项。
2. 在 AWS IoT Events 控制台的左上角，选择并展开导航窗格。
3. 在左侧导航窗格中，选择输入。
4. 在控制台的右上角，选择创建输入。
5. 提供独一无二的 InputName.
6. 可选-输入输入的描述。
7. 要上传 JSON **`input.json`** 文件，请在概述中选择您创建的文件创建一个 JSON 输入文件。选择输入属性随即出现，其中包含您输入的属性的列表。
8. 对于选择输入属性，选择要使用的属性，然后选择创建。在此示例中，我们选择motorid 和 sensorData.pressure。
9. 可选-在输入中添加相关标签。

> **Note**
>
> 您还可以在AWS IoT Events 控制台的探测器模型中创建其他输入。有关更多信息，请参阅 在探测器模型中创建输入 AWS IoT Events。

在探测器模型中创建输入 AWS IoT Events

中的探测器输入 AWS IoT Events 充当数据源和探测器模型之间的桥梁。探测器输入提供原始数据，为的事件检测和自动化功能提供支持 AWS IoT Events。学习配置探测器输入，以帮助您的模型准确响应物联网生态系统中的真实事件和条件。

本节介绍如何为探测器模型定义输入以接收遥测数据或消息。

为探测器模型定义输入

1. 打开 AWS IoT Events 控制台。
2. 在 AWS IoT Events 控制台中，选择创建探测器模型。
3. 选择 新建。

4. 选择创建输入。
5. 对于输入，输入可InputName选的描述，然后选择上传文件。在显示的对话框中，选择您在概述中创建的`input.json`文件创建一个 JSON 输入文件。
6. 对于选择输入属性，选择要使用的属性，然后选择创建。在这个例子中，我们选择了 m otorID 和 sens ordata.Pressure。

在中创建探测器模型 AWS IoT Events

在该主题中，您使用状态来定义一个探测器模型（您的设备或进程的模型）。

对于每种状态，您定义条件（布尔值）逻辑，该逻辑评估传入的输入以检测重要事件。当检测到事件时，它会更改状态并可以启动其他操作。这些事件称为过渡事件。

在您的状态下，您还可以定义在探测器进入或退出该状态或收到输入时可以运行操作的事件（这些事件被称为`OnEnter`、`OnExit` 和 `OnInput` 事件）。仅当事件的条件逻辑计算为 `true` 时，操作才会运行。

创建探测器模型

1. 已为您创建了第一个探测器状态。要对其进行修改，请在主编辑空间中选择标有 State_1 标签的圆圈。
2. 在状态窗格中，输入州名称 OnEnter，然后选择添加事件。
3. 在添加 OnEnter 事件页面上，输入事件名称和事件条件。在此示例中，输入 `true` 以表示当进入状态时，事件始终处于启动状态。
4. 在 事件操作 下面，选择 添加操作。
5. 在事件操作下，执行以下操作：

   a. 选择设置变量

   b. 对于变量运算，选择赋值。

   c. 对于变量名称，输入要设置的变量的名称。

   d. 对于变量值，输入值 **`0`**（零）。
6. 选择保存。

   可以在探测器模型中的任何事件中设置变量（赋值），例如您定义的变量。只有在探测器达到状态并运行定义或设置变量的操作后，才能引用变量的值（例如，在事件的条件逻辑中）。
7. 在“状态” 窗格中，选择“状态”旁边的 X 以返回到探测器模型调色板。

8. 要创建第二个探测器状态，请在探测器模型调色板中，选择状态并将其拖到主编辑空间中。这将创建一个名为 untitled_state_1 的状态。
9. 在第一个状态（正常）下暂停。状态的周长上会出现一个箭头。
10. 单击箭头并将其从第一个状态拖动到第二个状态。将出现从第一个状态到第二个状态的定向线（标记为“无标题”）。
11. 选择“无标题” 行。在过渡事件窗格中，输入事件名称和事件触发逻辑。
12. 在过渡事件窗格中，选择添加操作。
13. 在添加过渡事件操作窗格上，选择添加操作。
14. 在选择操作中，选择设置变量。
    a. 对于变量运算，选择赋值。
    b. 对于 变量名称，输入变量的名称。
    c. 对于 赋值，输入值，例如： `$variable.pressureThresholdBreached + 3`。
    d. 选择保存。
15. 选择第二个状态 untitled_state_1。
16. 在状态窗格中，输入状态名称，然后在On Enter中选择添加事件。
17. 在添加 OnEnter 事件页面上，输入事件名称和事件条件。选择添加操作。
18. 在选择操作中，选择 发送 SNS 消息。
    a. 对于 SNS 主题，请输入 Amazon SNS 主题的目标 ARN。
    b. 选择保存。
19. 继续在示例中添加事件。
    a. 对于 OnInput，选择添加事件，然后输入并保存以下事件信息。

```
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. 对于 OnInput，选择添加事件，然后输入并保存以下事件信息。

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

c. 对于 OnExit，选择添加事件，然后使用您创建的 Amazon SNS 主题的 ARN 输入并保存以下事件信息。

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

20. 暂停第二个状态（危险）。该状态的周长上会出现一个箭头
21. 单击箭头并将其从第二个状态拖动到第一个状态。将出现一条标有无标题标签的定向线。
22. 选择无标题行，然后在过渡事件窗格中，使用以下信息输入事件名称和事件触发逻辑。

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

有关我们为何在触发器逻辑中测试 `$input` 值和 `$variable` 值的更多信息，请参阅 AWS IoT Events 探测器型号限制和限制 中变量值可用性的条目。

23. 选择开始状态。默认情况下，此状态是在您创建探测器模型时创建的）。在 开始窗格中，选择目标状态（例如，正常）。
24. 接下来，配置您的探测器模型以监听输入。选择右上角的发布。
25. 在发布探测器模型页面，执行以下操作。

a. 输入探测器模型名称、描述和角色名称。将为您创建此角色。

b. 选择为每个唯一键值创建探测器。要创建和使用自己的角色，请按照 为设置权限 AWS IoT Events 中的步骤操作，并在此处将其作为角色输入。

26. 对于探测器创建密钥，请选择您之前定义的输入的其中一个属性的名称。您选择作为探测器创建密钥的属性必须存在于每个消息输入中，并且对于每台发送消息的设备都必须是唯一的。此示例使用 motorid 属性。
27. 选择保存并发布。

> **Note**
>
> 为给定探测器模型创建的唯一探测器的数量取决于发送的输入消息。创建探测器模型时，会从输入属性中选择一个密钥。此密钥决定要使用哪个探测器实例。如果以前从未见过密钥（对于此探测器模型），则会创建一个新的探测器实例。如果以前见过密钥，我们使用与该密钥值相对应的现有探测器实例。

您可以制作探测器模型定义的备份副本（采用 JSON 格式），重新创建或更新探测器模型，也可以将其用作模板来创建另一个探测器模型。

您可以从控制台或使用以下 CLI 命令执行此操作。如有必要，请更改探测器模型的名称，使其与您在上一步中发布时使用的名称相匹配。

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

这将创建一个文件 (`motorDetectorModel.json`)，其内容类似于以下内容。

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

发送输入以测试探测器模型 AWS IoT Events

有几种方法可以接收遥测数据 AWS IoT Events （请参阅支持在中接收数据和触发操作的操作 AWS IoT Events）。本主题介绍如何在 AWS IoT 控制台中创建 AWS IoT 规则，将消息作为输入转发给

AWS IoT Events 探测器。您可以使用 AWS IoT 控制台的 MQTT 客户端发送测试消息。您可以使用此方法获取遥测数据，了解您的设备 AWS IoT Events 何时能够使用消息代理发送 MQTT 消息。 AWS IoT

发送输入以测试探测器模型

1. 打开 AWS IoT Core 控制台。在左侧导航窗格的管理下，选择消息路由，然后选择规则。
2. 选择右上角的创建规则。
3. 在 创建规则页面上，完成以下步骤：

   1. 步骤 1：指定规则属性。填写以下字段：
      - 规则名称。输入规则的名称，例如 `MyIoTEventsRule`。

        > **Note**
        > 不要使用空格。

      - 规则描述。这是可选的。
      - 选择 下一步。
   2. 步骤 2：配置 SQL 语句。填写以下字段：
      - SQL 版本。从该列表中选择适当的选项。
      - SQL 语句。输入 **`SELECT *, topic(2) as motorid FROM 'motors/+/status'`**。

      选择下一步。
   3. 第 3 步：附加规则操作。在规则操作部分，完成以下操作：
      - 操作 1。选择 IoT 事件。显示以下字段：
        a. 输入名称。从该列表中选择适当的选项。如果您的输入未显示，请选择 刷新。

        要创建新输入，请选择创建 IoT 事件输入。填写以下字段：
        - 输入名称。输入 `PressureInput`。
        - 描述。这是可选的。
        - 上传一个 JSON 文件。上传您的 JSON 文件的副本。如果您没有文件，则此屏幕上会有一个指向示例文件的链接。该代码包括：

```
{
  "motorid": "Fulton-A32",
```

```
  "sensorData": {
    "pressure": 23,
    "temperature": 47
  }
}
```

- 选择输入属性。选择相应的选项。
- 标签。这是可选的。

选择 创建。

返回到创建规则屏幕并刷新输入名称字段。选择您创建的输入。

b. 批量模式。这是可选的。如果负载是一组消息，请选择此选项。

c. 消息 ID。您可以自由选择，但我们建议您这样做。

d. IAM 角色。从该列表中选择适当的角色。如果未列出该角色，请选择创建新角色。

键入角色名称，然后选择创建。

要添加其他规则，请选择 添加规则操作

- 错误操作。此部分是可选的。要添加操作，请选择添加错误操作，然后从列表中选择相应的操作。

填写显示的字段。

- 选择下一步。

4. 步骤 4：审核和创建 检查屏幕上的信息，然后选择 创建。

4. 在左导航窗格上的测试下，选择 MQTT 测试客户端。

5. 选择发布到主题。填写以下字段：

- 主题名称。输入用于标识消息的名称，例如 `motors/Fulton-A32/status`。
- 消息负载。输入以下信息：

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

> **Note**
>
> 每次发布新消息时都要更改 `messageId`。

6. 对于发布，请保持主题不变，但将负载中的 `"pressure"` 更改为大于您在探测器模型中指定的阈值的值（例如 **85**）。
7. 选择 发布。

您创建的探测器实例会生成并向您发送一条 Amazon SNS 消息。继续发送压力读数高于或低于压力阈值（本示例为 70）的消息，以查看探测器的运行情况。

在此示例中，您必须发送三条压力读数低于阈值的消息才能转换回正常状态，并收到一条表明超压状况已清除的 Amazon SNS 消息。回到正常状态后，一条压力读数超过极限的消息会导致探测器进入危险状态并发送一条表明该状况的 Amazon SNS 消息。

现在，您已经创建了一个简单的输入和探测器模型，请尝试以下操作。

- 在控制台上查看更多探测器模型示例（模板）。
- 按照中的步骤使用 CLI 为两种状态创建 AWS IoT Events 检测器使用创建输入和探测器模型 AWS CLI
- 了解事件中使用的 用于筛选、转换和处理事件数据的表达式 的详细信息。
- 了解 支持在中接收数据和触发操作的操作 AWS IoT Events。
- 如果某些东西不起作用，请参阅 故障排除 AWS IoT Events。

以下方面的最佳实践 AWS IoT Events

按照这些最佳实践可以从 AWS IoT Events获得最大的好处。

主题

在开发 AWS IoT Events 探测器模型时启用 Amazon CloudWatch 日志记录

Amazon 会实时 CloudWatch 监控您的 AWS 资源和您运行 AWS 的应用程序。借 CloudWatch助，您可以获得对资源使用情况、应用程序性能和运行状况的全系统可见性。当你开发或调试 AWS IoT Events 探测器模型时， CloudWatch可以帮助你知道 AWS IoT Events 正在做什么，以及它遇到的任何错误。

要启用 CloudWatch

1. 如果您还没有，请按照中的为设置权限 AWS IoT Events步骤创建具有附加策略的角色，该策略授予创建和管理 CloudWatch 日志的权限 AWS IoT Events。
2. 转到 AWS IoT Events 控制台。
3. 在导航窗格中，选择设置。
4. 在“设置”页面上，选择“编辑”。
5. 在 “编辑日志选项” 页面的 “日志选项” 部分，执行以下操作：
   a. 在 “详细程度” 中，选择一个选项。
   b. 在“选择角色”中，选择一个具有足够权限的角色来执行所选的日志操作。
   c. （可选）如果您选择了 “调试” 作为详细级别，则可以通过执行以下操作来添加调试目标：
      i. 在 “调试目标” 下，选择 “添加模型选项”。
      ii. 输入探测器型号名称和（可选）KeyValue以指定要记录的探测器型号和特定的探测器（实例）。
6. 选择更新。

您的日志选项已成功更新。

定期发布以在 AWS IoT Events 控制台中工作时保存您的探测器模型

使用 AWS IoT Events 控制台时，正在进行的作品会保存在本地浏览器中。但是，必须选择“发布”才能将探测器模型保存到 AWS IoT Events。在您发布探测器模型后，您发布的工作将在您用来访问帐户的任何浏览器中显示。

> **Note**
>
> 如果您不发布工作，则不会将其保存。发布探测器模型后，便无法再更改其名称。但是，您可以继续修改其定义。

AWS IoT Events 用例教程

AWS IoT Events 教程提供了一系列过程 AWS IoT Events，涵盖了从基本设置到更具体的用例的各个方面。每个教程都展示了实际场景的示例，可帮助您培养创建探测器模型、配置输入、设置操作以及与其他 AWS 服务集成以创建强大的 IoT 解决方案的实际技能。

本章将向您介绍：

- 帮助您决定纳入探测器模型的状态，确定您是需要一个探测器实例还是多个。
- 按照使用 AWS CLI的示例操作。
- 创建输入值，以接收来自设备的遥测数据。创建模型，以监视和报告数据发送设备的状态。
- 查看对输入、探测器模型和 AWS IoT Events 服务的限制。
- 了解探测器模型的更复杂示例，包含有注释。

主题

AWS IoT Events 用于监控您的物联网设备

您可以使用监控 AWS IoT Events 您的设备或进程，并根据重大事件采取行动。为此，请按照以下基本步骤操作：

创建输入

您必须通过一定的方法将设备和进程中的遥测数据导入 AWS IoT Events。若要执行此操作，请将消息作为输入发送到 AWS IoT Events。您可通过以下几种方式发送作为输入的消息：

- 使用该 BatchPutMessage操作。
- 为AWS IoT Core 规则引擎定义一个`iotEvents`规则操作。规则操作会将您输入的消息数据转发至 AWS IoT Events。
- 在中 AWS IoT Analytics，使用CreateDataset操作创建数据集`contentDeliveryRules`。这些规则指定了自动将数据集内容发送到哪个 AWS IoT Events 输入。

- 在 AWS IoT Events 探测器模型或事件中定义 IoteVents 动作。onInput onExit transitionEvents对于探测器模型实例和操作启动事件信息，其将作为输入反馈至系统，名称为您指定的名称。

在设备开始以这种方式发送数据之前，您必须定义一个或多个输入。为此，请为每个输入指定一个名称，并指定输入监视传入消息数据中的哪些字段。 AWS IoT Events 以 JSON 有效载荷的形式接收来自多个来源的输入。每项输入可单独操作，或与其他输入组合操作，以检测更复杂的事件。

创建探测器模型

使用状态创建一个探测器模型（您的设备或进程的模型）。对于每种状态，您定义条件（布尔值）逻辑，该逻辑评估传入的输入以检测重要事件。当检测到事件时，它可以使用其他 AWS 服务更改状态或启动自定义或预定义的操作。您可以定义其他事件，这些事件将在进入或退出某个状态以及满足某个条件（可选）时发起操作。

在本教程中，您将学习在模型进入或退出特定状态时，如何发送作为操作的 Amazon SNS 消息。

监视设备或进程

如果您正在监视多个设备或进程，则可以在每个输入中指定一个字段，用于识别输入来自的特定设备或进程。（请参见CreateDetectorModel中的 key 字段。）当识别出新设备时（在key识别的输入字段中看到一个新值），就会创建一个探测器。（每个探测器就是一个探测器模型实例。）然后，新的探测器继续响应来自该设备的输入，直到其探测器模型被更新或删除。

如果您正在监视单个进程（即使多个设备或子进程正在发送输入），则无需指定唯一的标识 key 字段。在这种情况下，当第一次输入到达时，将创建单个探测器（实例）。

将消息作为输入发送至您的探测器模型

有几种方法可以将来自设备或进程的消息作为输入发送到 AWS IoT Events 检测器，而无需对消息进行其他格式化。在本教程中，您将使用 AWS IoT 控制台为将消息数据转发到 AWS IoT Events的 AWS IoT Core 规则引擎编写AWS IoT Events 操作规则。为此，您需要按名称识别输入。然后，您继续使用 AWS IoT 控制台生成一些消息，这些消息将作为输入转发给 AWS IoT Events。

您如何知道在探测器模型中需要什么状态？

若要确定您的探测器模型应有的状态，首先应确定要采取何种操作。例如，如果您的汽车使用汽油，您可以在开始行程时查看燃油表，了解是否需要加油。这种情况下的操作是：告知驾驶员“去加油”。您的探测器模型需要两种状态：“汽车不需要加油”和“汽车确实需要加油”。通常，您要为每个可能的操作定义一个状态，再为不需要的操作定义一个状态。这在操作本身比较复杂的情况下也有效。例如，您可能想查找并纳入有关最近加油站或最便宜油价的信息，但是当您发送“去加油”的消息时，您会这样做。

要决定接下来要进入哪种状态，请查看输入。输入包含决定您应该处于何种状态所需的信息。要创建一个输入，请在设备或进程发送的消息中选择一个或多个有助于您决定的字段。在此示例中，您需要一个输入，告诉您当前的燃油油位（“满油百分比”）。也许汽车正在向您发送多条不同的消息，每条消息都包含多个不同的字段。若要创建此输入，您必须选择该消息以及用于报告当前油量的字段。为简化操作，您可对行程长度（“到目的地的距离”）进行硬编码；您可使用平均行程长度。您将根据输入执行一些计算（该加注百分比可转换为多少加仑？平均行程长度是否大于您的行驶里程（考虑到您拥有的加仑数和平均“每加仑英里数”）。您可以执行上述计算，并在事件中发送消息。

目前您已有两个状态和一个输入。您需要事件处于第一种状态，即根据输入执行计算，并决定是否转至第二种状态。此为过渡事件（`transitionEvents` 在状态的 `onInput` 事件列表中。当收到第一状态的输入 时，事件执行过渡到第二种状态，前提是满足事件的 `condition`。）当到达第二种状态时，您会在进入该状态后立即发送消息。（您使用 `onEnter` 事件。进入第二种状态时，此事件将发送消息。无需等待其他输入。）还有其他类型事件，但这个简单的示例就包含了所有必要内容。

其他类型的事件是 `onExit` 和 `onInput`。收到输入并满足条件后，`onInput` 事件就会执行指定行动。当操作退出其当前状态并且满足条件时，`onExit` 事件将执行指定操作。

您是否漏掉了什么？ 是的，如何回到第一种“车不需要加油”的状态？ 油箱加满后，输入显示油箱已满。在第二种状态下，您需要过渡事件回到第一种状态，该状态在收到输入时发生（在第二个状态的 `onInput:` 事件中）。如果计算结果显示您的油量足以让您前往想去的地方，应过渡回第一种状态。

这就是基础理论。部分探测器模型通过添加可反映重要输入（而不仅仅是可能的操作）的状态而变得更加复杂。例如，在追踪温度的探测器模型中，可能有三种状态：“正常”状态、“过热”状态和“潜在问题”状态。当温度升高至一定水平以上，但还没有变得过热时，就会过渡到“潜在问题”状态。如果此温度保持 15 分钟以上，就不需要发送警报。如果在此之前温度恢复正常，探测器将恢复至正常状态。为了谨慎起见，如果计时器到达十五分钟时限，探测器会过渡到过热状态并发送警报。您可以使用变量和一组更复杂的事件条件，完成同样的操作。但实际上，使用另一种状态存储计算结果通常更容易。

您如何知道自己是需要探测器的一个实例还是多个？

为决定需要的实例数量，请问自己“你想知道什么？” 假设您想知道今天的天气状况。下雨了吗（状态）？ 您需要带雨伞（行动）吗？ 您有一个报告温度的传感器、一个报告湿度的传感器，以及报告气压、风速、风向、降水量的其他传感器。但是您必须同时监视所有这些传感器，以确定天气状态（雨、雪、阴天、晴天）和要采取的适当行动（拿雨伞或涂防晒霜）。尽管传感器数量众多，但您仍需要通过探测器实例监视天气状态，并告知自己要采取的行动。

但是，如果您是所在地区的天气预报员，您可能需要多个这样的传感器阵列实例，它们位于该地区的不同位置。每个位置的人都需要了解所在位置的天气情况。在此情况下，您需要有多个探测器实例。每个位置的每个传感器报告的数据必须包含已指定为 `key` 字段的字段。该字段让 AWS IoT Events 可以为

该区域创建探测器实例，然后在它到达时，继续将这些信息路由至该探测器实例。不再担心头发淋湿或鼻子晒伤！

本质上，如果您要监视一种情况（一个进程或位置），则需要一个探测器实例。如果您要监视多种情况（位置、进程），则需要多个探测器实例。

使用 CLI 为两种状态创建 AWS IoT Events 检测器

在此示例中，我们调用 usin AWS IoT Events APIs g AWS CLI 命令来创建一个探测器，该探测器对发动机的两种状态进行建模：正常状态和超压状态。

当引擎中测得的压力超过一定阈值时，模型会过渡至超压状态，并发送 Amazon Simple Notification Service（Amazon SNS）消息，提醒技术人员注意该情况。当压力降至连续三个压力读数的阈值以下时，模型将恢复至正常状态，并发送另一条 Amazon SNS 消息，以确认该状况已消除。我们需要获得低于压力阈值的三个连续读数，以消除在非线性恢复阶段或一次性异常恢复读数的情况下可能出现的过压/正常消息卡顿现象。

下文概述了探测器创建步骤。

创建输入。

要监视您的设备和流程，它们必须具有将遥测数据导入 AWS IoT Events的方法。这是通过向发送消息作为输入来完成的 AWS IoT Events。有几种方式可以实现：

- 使用该 BatchPutMessage操作。此方法很简单，但要求您的设备或进程能够 AWS IoT Events 通过 SDK 或 AWS CLI.
- 在中 AWS IoT Core，为将您的消息数据转发到 AWS IoT Events的 AWS IoT Core 规则引擎编写一条AWS IoT Events 操作规则。按名称识别输入。如果您的设备或进程可以或已经通过发送消息，请使用此方法 AWS IoT Core。这种方法通常对设备的计算能力要求不高。
- 在中 AWS IoT Analytics，使用 CreateDataset操作创建数据集`contentDeliveryRules`，该数据集用于指定 AWS IoT Events 输入，数据集内容将在其中自动发送。如果您想根据在 AWS IoT Analytics中汇总或分析的数据控制您的设备或进程，请使用此方法。

在您的设备以这种方式发送数据之前，您必须定义一个或多个输入。为此，请为每个输入赋予一个名称，并指定输入要监视传入消息数据中的哪些字段。

创建探测器模型

使用状态创建一个探测器模型（您的设备或进程的模型）。对于每种状态，请定义条件（布尔值）逻辑，该逻辑评估传入的输入以检测重要事件。当检测到事件时，它可以使用其他 AWS 服务更改

状态或启动自定义或预定义的操作。您可以定义其他事件，这些事件将在进入或退出某个状态以及满足某个条件（可选）时发起操作。

监视多个设备或进程

如果您正在监视多个设备或进程，并且想要单独追踪每个设备或进程，请在每个输入中指定一个字段，可用于识别输入来自的特定设备或进程。请参见`CreateDetectorModel`中的 `key` 字段。当识别出新设备时（在`key`识别的输入字段中看到一个新值），就会创建一个探测器实例。新的探测器实例会继续响应来自特定设备的输入，直至探测器模型更新或被删除。您的唯一探测器（实例）数量与输入`key`字段的唯一值相同。

监视单个设备或进程

如果您正在监视单个进程（即使多个设备或子进程正在发送输入），则无需指定唯一的标识 `key` 字段。在这种情况下，当第一次输入到达时，将创建单个探测器（实例）。例如，一个房屋的每个房间都安装有温度传感器，但是只有一台暖通空调用于为整个房子供暖和制冷。因此，即使每个房间的占用者都希望他们的投票（输入）占上风，您也只能将其作为单个进程进行控制。

将来自设备或进程的消息作为输入发送至探测器模型

我们描述了从设备或进程发送消息作为输入到 AWS IoT Events 探测器的输入的几种方法。创建输入并构建探测器模型后，您就可以开始发送数据。

> **Note**
>
> 创建探测器模型或更新现有探测器模型后，新的或更新的探测器模型需要几分钟才能开始接收消息和创建探测器（实例）。如果探测器模型已更新，则在此期间您可能会继续看到基于先前版本的行为。

主题

创建用于捕获设备数据的 AWS IoT Events 输入

为设置输入时 AWS IoT Events，您可以利用 AWS CLI 来定义设备如何传输传感器数据。例如，如果您的设备发送带有电机标识符和传感器读数的 JSON 格式的消息，则可以通过创建映射消息中特定属

性（例如压力和电机 ID）的输入来捕获这些数据。该过程首先在 JSON 文件中定义输入，指定相关的数据点，然后使用注册输入 AWS IoT Events。 AWS CLI 这使得 AWS IoT 能够根据实时传感器数据监控和响应关键条件。

例如，假设您的设备按以下格式发送消息。

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

您可以使用以下 AWS CLI 命令创建用于捕获`pressure`数据和`motorid`（用于标识发送消息的特定设备）的输入。

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

`pressureInput.json` 文件包含以下内容。

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

创建自己的输入时，切记先从您的设备或进程收集 JSON 文件形式的示例消息。您可将此消息用于创建来自控制台或 CLI 的输入。

创建用于表示设备状态的探测器模型 AWS IoT Events

在创建用于捕获设备数据的 AWS IoT Events 输入，您根据电机压力数据报告消息，创建了`input`。继续举例，有一个响应电机过压事件的探测器模型。

您可以创建两种状态：“Normal”和“Dangerous”。创建后，每种探测器（实例）都会进入“Normal”状态。当具有唯一key值的“motorid”输入到达时，可创建实例。

如果探测器实例收到的压力读数为 70 或以上，它将进入“Dangerous”状态并发送 Amazon SNS 消息作为警告。如果连续三次输入的压力读数恢复正常（小于 70），探测器将返回“Normal”状态，并发送另一条 Amazon SNS 消息作为警报解除信号。

此示例检测器模型假设您创建了两个 Amazon SNS 主题，其亚马逊资源名称 (ARNs) 在定义中显示为"targetArn": "arn:aws:sns:us-east-1:123456789012:underPressureAction"和。"targetArn": "arn:aws:sns:us-east-1:123456789012:pressureClearedAction"

有关更多信息，请参阅《亚马逊简单通知服务开发者指南》，更具体地说，请参阅《亚马逊简单通知服务 API 参考》中的CreateTopic操作文档。

此示例还假设您已创建具有相应权限的 AWS Identity and Access Management (IAM) 角色。该角色的 ARN 在探测器模型定义中显示为"roleArn": "arn:aws:iam::123456789012:role/IoTEventsRole"。按为设置权限 AWS IoT Events中的步骤创建此角色，将角色的 ARN 复制到探测器模型定义的适当位置。

您可以使用以下 AWS CLI 命令创建探测器模型。

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

"motorDetectorModel.json" 文件包含以下内容。

```
{
  "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"
}
```

将消息作为输入发送给探测器 AWS IoT Events

现在您已定义了一个输入，用于识别从设备发送的消息中的重要字段（参见创建用于捕获设备数据的 AWS IoT Events 输入）。在上一节中，您创建了一个detector model，它可以响应电机的过压事件（参见创建用于表示设备状态的探测器模型 AWS IoT Events）。

要完成此示例，请将来自设备（此例中为已安装 AWS CLI 的计算机）的消息作为输入发送至探测器。

> **Note**
>
> 创建探测器模型或更新现有探测器模型时，新的或更新的探测器模型需要几分钟才能开始接收消息和创建探测器（实例）。如果您更新探测器模型，则在此期间，您可能会继续看到基于先前版本的行为。

使用以下 AWS CLI 命令发送包含超出阈值的数据的消息。

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

文件“highPressureMessage.json”包含以下内容。

```
{
  "messages": [
```

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

您必须更改每条已发送消息中的 messageId。如果您不对其进行更改，则 AWS IoT Events 系统会对消息进行重复数据删除。 AWS IoT Events 如果一条消息与最近五分钟内发送的另一封邮件messageID相同，则忽略该消息。

此时，将创建一个探测器（实例）以监视电机事件"Fulton-A32"。该探测器在创建时进入 "Normal" 状态。但是，由于我们发送的压力值高于阈值，因此它会立即转换为 "Dangerous" 状态。在此过程中，探测器会向 ARN 为arn:aws:sns:us-east-1:123456789012:underPressureAction的 Amazon SNS 端点发送消息。

运行以下 AWS CLI 命令以发送包含低于压力阈值的数据的消息。

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

normalPressureMessage.json 文件包含以下内容。

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

每次当您在五分钟内调用BatchPutMessage命令时，都必须更改文件中的messageId。再发送该信息两次。消息发送三次后，电机“Fulton-A32”的探测器（实例）会向 Amazon SNS 端点"arn:aws:sns:us-east-1:123456789012:pressureClearedAction"发送一条消息并重新进入"Normal"状态。

> **Note**
>
> 您可以使用BatchPutMessage同时发送多条消息。但是，不保证处理这些消息的顺序。为确保按顺序处理消息（输入），请每次发送一条消息，然后等待每次调用 API 时成功响应。

以下是本节所述的由探测器模型示例创建的 SNS 消息有效载荷示例。

事件“违反压力阈值”

```
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"
}
```

事件“恢复正常压力”

```
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"
}
```

如果您定义了任何计时器，则其当前状态也会在 SNS 消息有效载荷中显示。

消息有效载荷包含消息发送时（也就是运行 SNS 操作时）的探测器（实例）状态信息。您可以使用 https://docs.aws.amazon.com/iotevents/latest/apireference/API_iotevents-data_DescribeDetector.html 操作获取有关探测器状态的类似信息。

AWS IoT Events 探测器型号限制和限制

创建探测器模型时，需要考虑以下事项。

如何使用 **actions** 字段

`actions` 字段是一个对象列表。您可以有多个对象，但每个对象仅有一次操作。

Example

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

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

如何使用 **condition** 字段

condition 是 transitionEvents 的必填项，在其他情况下为可选。

如果condition 字段不存在，则等同于 "condition": true。

条件表达式的计算结果应为布尔值。如果结果非布尔值，则它等同于 false，且不会发起actions或转换为事件中指定的nextState。

变量值的可用性

默认情况下，如果在事件中设置了变量值，则其新值不可用或无法用于评估同组其他事件的条件。新值不可用，也无法用于相同onInput、onEnter 或 onExit字段的事件条件。

在探测器模型定义中设置 evaluationMethod 参数以更改此行为。当 evaluationMethod 设置为 SERIAL 时，将更新变量，并按照事件的定义顺序评估事件条件。否则，当evaluationMethod设置为BATCH或默认为它时，状态变量会更新，并且只有在评估了所有事件条件之后才会执行状态内事件。

在 "Dangerous" 状态的 onInput 字段中，如果满足条件（当前输入的压力小于或等于 70 时），则"Pressure Okay"事件中的"$variable.pressureThresholdBreached"减一。

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

```
            }
```

当"Normal"达到 0 时（也就是说，当探测器连续接收到三个小于等于 70 的压力读数时），探测器应该恢复到"$variable.pressureThresholdBreached"状态。"BackToNormal"中的transitionEvents事件必须测试"$variable.pressureThresholdBreached"小于等于 1（非 0），并且再次验证"$input.PressureInput.sensorData.pressure"提供的当前值是小于或等于 70。

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

否则，如果条件仅测试变量的值，则两个正常读数后跟一个超压读数将满足条件并过渡回"Normal"状态。条件是查看在上次处理输入时给出的值"$variable.pressureThresholdBreached"。"Overpressurized"事件中的变量值重置为 3，但是切记此新值现在不适用于任何condition。

默认情况下，每次控件输入 onInput 字段时，condition只能获取变量在开始处理输入时的值，然后才会通过onInput中指定的任何操作更改。onEnter 和 onExit也是如此。当我们进入或退出该状态时，对变量所做的任何更改都不适用于相同 onEnter 或 onExit 字段中指定的其他条件。

更新探测器模型时延迟

如果您更新、删除和重新创建探测器模型（参见 UpdateDetectorModel），则在删除所有生成的探测器（实例）并使用新模型重新创建探测器之前，会有一些延迟。新探测器模型生效且新输入到达后，将重新创建它们。在此期间，输入可能会继续由先前版本的探测器模型生成的探测器处理。在此期间，您可能会继续收到由先前探测器模型定义的警报。

输入键中的空格

输入键中允许使用空格，但是无论是在输入属性的定义中，还是在表达式中引用键值时，都必须用反引号括住对键的引用。例如，假定如下消息有效负载：

```
{
```

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

使用以下内容定义输入。

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

在条件表达式中，您还必须使用反引号来引用任何此类键的值。

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

一个有评论的例子：暖通空调温度控制 AWS IoT Events

以下是含内联注释（使 JSON 无效）的 JSON 文件示例 这些示例的完整版本（无注释）可见示例：将 HVAC 温度控制与 AWS IoT Events。

此示例实现了一个恒温器控制模型，该模型使您能够执行以下操作。

- 仅定义一个可用于监视和控制多个区域的探测器模型。为每个区域创建一个探测器实例。
- 从每个控制区域的多个传感器摄取温度数据。
- 更改某个区域的温度设定点。
- 为每个区域设定操作参数，并在使用实例时重置这些参数。
- 动态地添加或删除某个区域的传感器。
- 指定最短运行时间以供暖和制冷装置。
- 拒绝异常传感器读数。

- 定义紧急设定点，如果任何传感器报告的温度高于或低于给定阈值，该设定点会立即启动供暖或制冷。
- 报告异常读数及温度峰值。

主题

中探测器模型的输入定义 AWS IoT Events

我们想创建一个探测器模型，可用于监测和控制多个不同区域的温度。每个区域可以有多个温度报告传感器。我们假定每个区域均配备一台供暖装置和一台制冷装置，其可以打开或关闭以控制该区域的温度。每个区域都由探测器实例控制。

由于我们监视和控制的不同区域可能具有不同的特征，需要不同的控制参数，因此我们定义`'seedTemperatureInput'`，以便为每个区域提供这些参数。当我们将其中一条输入消息发送至AWS IoT Events时，将创建一个新的探测器模型实例，其中包含我们要在该区域使用的参数。以下是该输入的定义。

CLI 命令：

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

`seedInput.json` 文件：

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

响应：

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

注意

- 为任何消息中收到的每个唯一'areaId'创建新探测器实例。请参见'areaDetectorModel'定义中的'key'字段。
- 激活该区域的供暖或制冷装置前，平均温度可能与'desiredTemperature'不同，可能会相差'allowedError'。
- 如果任何传感器报告的温度高于'rangeHigh'，则探测器会报告峰值并立即启动制冷装置。
- 如果任何传感器报告的温度低于'rangeLow'，则探测器会报告峰值并立即启动供暖装置。
- 如果任何传感器报告的温度高于 'anomalousHigh' 或低于 'anomalousLow'，则探测器会报告异常传感器读数，但会忽略报告的温度读数。
- 'sensorCount' 告知探测器该区域报告传感器的数量。探测器为其接收到的每个温度读数给出适当的权重系数，从而计算该区域的平均温度。因此，探测器不必追踪每个传感器报告的内容，并且可

以根据需要动态更改传感器数量。但是，如果单个传感器离线，探测器将无法知道或考虑这一点。我们建议您创建另一个专门用于监视每个传感器连接状态的探测器模型。使用两个互补的探测器模型可简化两者的设计。

- 'noDelay' 值可能是 true 或 false。供暖或制冷装置开启后，应保持开启状态达到一定的最短时间，以保护设备的完整性并延长工作寿命。如果'noDelay'设置为 false，则探测器实例会在关闭制冷和供暖装置之前强制延迟，以确保它们的运行达到最短时间。延迟秒数已在探测器模型定义中硬编码，因为我们无法使用变量值来设置计时器。

'temperatureInput' 用于将传感器数据传输至探测器实例。

CLI 命令：

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

temperatureInput.json 文件：

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

响应：

```
{
    "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."
    }
```

```
}
```

注意

- 示例探测器实例不使用'sensorId'来直接控制或监视传感器。它会自动传递至探测器实例发送的通知中。从此处，它可用于识别出现故障的传感器（例如，定期发送异常读数的传感器可能即将失效）或已离线的传感器（当它被用作监视设备重要特征的额外探测器模型的输入时）。如果某个区域的读数经常与平均值不同，'sensorId'还可以帮助识别该区域的温暖或寒冷区域。
- 'areaId' 用于将传感器的数据路由至相应的探测器实例。为任何消息中收到的每个唯一'areaId'创建探测器实例。请参见'areaDetectorModel'定义中的'key'字段。

创建探 AWS IoT Events 测器模型定义

'areaDetectorModel' 示例包含内联注释。

CLI 命令：

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

areaDetectorModel.json 文件：

```
{
  "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"
}
```

响应：

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

BatchUpdateDetector 用于更新 AWS IoT Events 探测器模型

您可以使用BatchUpdateDetector 操作将探测器实例置于已知状态，包括计时器和变量值。在以下示例中，BatchUpdateDetector 操作会重置处于温度监视和控制之下的区域的操作参数。通过此操作，您无需删除、重新创建或更新探测器模型。

CLI 命令：

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

areaDM.BUD.json 文件：

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

响应：

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

BatchPutMessage 用于中的输入 AWS IoT Events

Example 1

使用 BatchPutMessage 操作发送一条"seedTemperatureInput"消息，为受温度控制和监视的给定区域设置操作参数。任何收到的带有 AWS IoT Events 新消息的消息都会"areaId"导致创建新的探测器实例。但是，在收到新区域的"idle"消息前，新的探测器实例不会将状态更改为"seedTemperatureInput"，也不会开始监视温度、控制供暖或制冷。

CLI 命令：

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

seedExample.json 文件：

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

响应：

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

Example

2

使用 BatchPutMessage 操作发送 "temperatureInput" 消息，以报告给定控制和监视区域内传感器的温度传感器数据。

CLI 命令：

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

temperatureExample.json 文件：

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

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

响应：

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

Example 3

使用 BatchPutMessage 操作发送 "seedTemperatureInput" 消息，以更改给定区域的所需温度值。

CLI 命令：

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

seedSetDesiredTemp.json 文件：

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

响应：

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

在中提取 MQTT 消息 AWS IoT Events

如果您的传感器计算资源无法使用 "BatchPutMessage" API，但可以使用轻量级 MQTT 客户端将其数据发送到 AWS IoT Core 消息代理，则可以创建 AWS IoT Core 主题规则以将消息数据重定向到 AWS IoT Events 输入。以下是 AWS IoT Events 主题规则的定义，该规则采用 MQTT 主题中的"areaId"和"sensorId"输入字段，以及消息负载"sensorData.temperature""temp"字段中的字段，并将这些数据提取到我们的。 AWS IoT Events "temperatureInput"

CLI 命令：

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

seedSetDesiredTemp.json 文件：

```
{
  "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"
  }
}
```

响应：[无]

如果传感器通过以下有效负载发送了一条有关主题"update/temperature/Area51/03"的消息。

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

这会导致数据被摄取，就好 AWS IoT Events 像进行了以下 "BatchPutMessage" API 调用一样。

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

spoofExample.json 文件：

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

在中生成 Amazon SNS 消息 AWS IoT Events

以下是 "Area51" 探测器实例生成的 SNS 消息示例。

AWS IoT Events 可以与 Amazon SNS 集成，根据检测到的事件生成和发布通知。本节演示 AWS IoT Events 探测器实例（特别是 “Area51” 检测器）如何生成 Amazon SNS 消息。这些示例展示了 AWS IoT Events 探测器内各种状态和事件触发的 Amazon SNS 通知的结构和内容，说明了与 Amazon AWS IoT Events SNS 结合使用以进行实时警报和通信的强大功能。

```
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"}

在中配置 DescribeDetector API AWS IoT Events

中的 DescribeDetector API AWS IoT Events 允许您检索有关特定检测器实例的详细信息。此操作可深入了解探测器的当前状态、变量值和活动计时器。通过使用此 API，您可以监控 AWS IoT Events 探测器的实时状态，从而简化物联网事件处理工作流程的调试、分析和管理。

CLI 命令：

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

响应：

```
{
    "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"
    }
}
```

将 AWS IoT Core 规则引擎用于 AWS IoT Events

以下规则将 AWS IoT Core MQTT 消息作为影子更新请求消息重新发布。我们假设 AWS IoT Core 为由探测器模型控制的每个区域定义了加热装置和冷却装置。在此示例中，我们定义了名为 "Area51HeatingUnit" 和 "Area51CoolingUnit" 的元素。

CLI 命令：

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

ADMShadowCoolOffRule.json 文件：

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

```
}
```

响应：[空]

CLI 命令：

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

ADMShadowCoolOnRule.json 文件：

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

响应：[空]

CLI 命令：

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

ADMShadowHeatOffRule.json 文件：

```
{
```

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

响应：[空]

CLI 命令：

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

ADMShadowHeatOnRule.json 文件：

```
{
  "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"
        }
```

```
        }
      ]
    }
}
```

响应：[空]

支持在中接收数据和触发操作的操作 AWS IoT Events

AWS IoT Events 可以在检测到指定事件或过渡事件时触发操作。您可以定义内置操作以使用计时器或设置变量，或者将数据发送到其他 AWS 资源。了解如何配置和自定义这些操作，以创建对各种物联网事件的自动响应。

> **Note**
>
> 在探测器模型中定义行动时，可以对字符串数据类型的参数使用表达式。有关更多信息，请参阅表达式。

AWS IoT Events 支持以下允许您使用计时器或设置变量的操作：

- `setTimer` 创建计时器。
- `resetTimer` 重置计时器。
- `clearTimer` 删除计时器。
- `setVariable` 创建变量。

AWS IoT Events 支持以下允许您使用 AWS 服务的操作：

- `iotTopicPublish` 发布有关 MATT 主题的消息。
- `iotEvents` 将数据以输入值的形式发送至 AWS IoT Events 。
- `iotSiteWise` 将数据发送至 AWS IoT SiteWise中的资产属性。
- `dynamoDB` 向 Amazon DynamoDB 表发送数据。
- `dynamoDBv2` 向 Amazon DynamoDB 表发送数据。
- `firehose`将数据发送到 Amazon Data Firehose 流。
- `lambda` 调用 AWS Lambda 函数。
- `sns` 将数据作为推送通知发送。
- `sqs` 将数据发布至 Amazon SQS 队列。

使用 AWS IoT Events 内置计时器和可变动作

AWS IoT Events 支持以下允许您使用计时器或设置变量的操作：

- `setTimer` 创建计时器。
- `resetTimer` 重置计时器。
- `clearTimer` 删除计时器。
- `setVariable` 创建变量。

设置计时器操作

Set timer action

`setTimer` 操作可以让您创建一个持续时间以秒为单位的计时器。

More information (2)

创建计时器时，必须指定以下参数。

`timerName`

计时器的名称。

`durationExpression`

（可选）计时器的持续时间，以秒为单位。

持续时间表达式的计算结果向下舍入为最接近的整数。例如，如果您将计时器设置为 60.99 秒，则持续时间表达式的计算结果为 60 秒。

有关更多信息，请参阅《AWS IoT Events API Reference》中的 SetTimerAction。

重置计时器行动

Reset timer action

`resetTimer` 操作可以让您将计时器设置为此前计算的持续时间表达式结果。

More information (1)

重置计时器时，您必须指定以下参数。

`timerName`

计时器的名称。

AWS IoT Events 重置计时器时不会重新计算持续时间表达式。

有关更多信息，请参阅《AWS IoT Events API Reference》中的 ResetTimerAction。

清除计时器操作

Clear timer action

`clearTimer` 操作可以让您删除现有计时器。

More information (1)

删除计时器时，您必须指定以下参数。

`timerName`

计时器的名称。

有关更多信息，请参阅《AWS IoT Events API Reference》中的 ClearTimerAction。

设置变量操作

Set variable action

`setVariable` 操作可以让您创建具有指定值的变量。

More information (2)

创建变量时，您必须指定以下参数。

`variableName`

变量的名称。

`value`

变量的新值。

有关更多信息，请参阅《AWS IoT Events API Reference》中的 SetVariableAction。

AWS IoT Events 使用其他 AWS 服务

AWS IoT Events 支持以下允许您使用 AWS 服务的操作：

- `iotTopicPublish` 发布有关 MATT 主题的消息。
- `iotEvents` 将数据以输入值的形式发送至 AWS IoT Events 。
- `iotSiteWise` 将数据发送至 AWS IoT SiteWise中的资产属性。
- `dynamoDB` 向 Amazon DynamoDB 表发送数据。
- `dynamoDBv2` 向 Amazon DynamoDB 表发送数据。
- `firehose`将数据发送到 Amazon Data Firehose 流。
- `lambda` 调用 AWS Lambda 函数。
- `sns` 将数据作为推送通知发送。
- `sqs` 将数据发布至 Amazon SQS 队列。

> **Important**
>
> - 您必须为两者 AWS IoT Events 以及要使用的 AWS 服务选择相同的 AWS 区域。有关支持的区域列表，请参见 Amazon Web Services 一般参考 中的 AWS IoT Events 端点和配额。
> - 在为 AWS IoT Events 操作创建其他 AWS 资源时，必须使用相同的 AWS 区域。如果您切换 AWS 区域，则访问 AWS 资源可能会遇到问题。

默认情况下， AWS IoT Events 会为任何操作生成 JSON 格式的标准负载。此操作有效负载包含有关探测器模型实例和触发操作的事件的信息的所有属性/值对。要配置操作负载，您可使用内容表达式。有关更多信息，请参阅用于筛选、转换和处理事件数据的表达式和AWS IoT Events API 参考中的有效负载数据。

AWS IoT Core

IoT topic publish action

该 AWS IoT Core 操作允许您通过消息代理发布 MQTT AWS IoT 消息。有关支持的区域列表，请参见 Amazon Web Services 一般参考 中的 AWS IoT Core 端点和配额。

AWS IoT 消息代理通过从发布 AWS IoT 客户端向订阅客户端发送消息来连接客户端。有关更多信息，请参阅《AWS IoT 开发人员指南》中的设备通信协议。

More information (2)

发布 MQTT 消息时，您必须指定以下参数。

`mqttTopic`

接收该消息的 MQTT 主题。

您可以使用在探测器模型中创建的变量或输入，在运行时系统动态地定义 MQTT 主题名称。

`payload`

（可选）默认有效负载包含有关探测器模型实例和触发操作的事件的信息的所有属性/值对。此外，您还可以自定义负载。有关更多信息，请参阅AWS IoT Events 《API 参考》中的有效负载。

> **Note**
>
> 确保附加到您的 AWS IoT Events 服务角色的策略授予了该`iot:Publish`权限。有关更多信息，请参阅 的身份和访问管理 AWS IoT Events。

有关更多信息，请参阅《AWS IoT Events API Reference》中的 IotTopicPublishAction。

AWS IoT Events

IoT Events action

该 AWS IoT Events 操作允许您将数据 AWS IoT Events 作为输入发送到。有关支持的区域列表，请参见 Amazon Web Services 一般参考 中的 AWS IoT Events 端点和配额。

AWS IoT Events 允许您监控设备或设备群是否出现故障或运行变化，并在此类事件发生时触发操作。有关更多信息，请参阅什么是 AWS IoT Events？ 在《AWS IoT Events 开发人员指南》中。

More information (2)

向发送数据时 AWS IoT Events，必须指定以下参数。

`inputName`

接收数据的 AWS IoT Events 输入名称。

payload

（可选）默认有效负载包含有关探测器模型实例和触发操作的事件的信息的所有属性/值对。此外，您还可以自定义负载。有关更多信息，请参阅AWS IoT Events 《API 参考》中的有效负载。

> **Note**
>
> 确保附加到您的 AWS IoT Events 服务角色的策略授予了该`iotevents:BatchPutMessage`权限。有关更多信息，请参阅 的身份和访问管理 AWS IoT Events。

有关更多信息，请参阅《AWS IoT Events API Reference》中的 IotEventsAction。

AWS IoT SiteWise

IoT SiteWise action

该 AWS IoT SiteWise 操作允许您将数据发送到中的资产属性 AWS IoT SiteWise。有关支持的区域列表，请参见 Amazon Web Services 一般参考 中的 AWS IoT SiteWise 端点和配额。

AWS IoT SiteWise 是一项托管服务，可让您大规模收集、组织和分析来自工业设备的数据。有关更多信息，请参阅《AWS IoT SiteWise 用户指南》中的什么是 AWS IoT SiteWise？。

More information (11)

向中的资产属性发送数据时 AWS IoT SiteWise，必须指定以下参数。

> **Important**
>
> 要接收数据，您必须使用 AWS IoT SiteWise中的现有资产属性。
>
> - 如果您使用 AWS IoT Events 控制台，则必须指定`propertyAlias`以标识目标资产属性。
> - 如果使用 AWS CLI，则必须指定其中一个`propertyAlias`或两个`propertyId`，`assetId`然后才能标识目标资产属性。

有关更多信息，请参阅AWS IoT SiteWise 《用户指南》中的将工业数据流映射到资产属性。

propertyAlias

（可选）资产属性的别名。您也可以指定表达式。

assetId

（可选）包含指定属性的资产的 ID。您也可以指定表达式。

propertyId

（可选）资产的属性 ID。您也可以指定表达式。

entryId

（可选）此条目的唯一标识符。您可以使用条目 ID，跟踪在出现故障时哪些数据条目导致了错误。默认值为新的唯一标识符。您也可以指定表达式。

propertyValue

包含有关属性值详细信息的结构。

quality

（可选）资产属性值的质量。值必须为 GOOD、BAD 或 UNCERTAIN。您也可以指定表达式。

timestamp

（可选）包含时间戳信息的结构。如果您未指定此值 ，则默认为事件时间。

timeInSeconds

采用 Unix 纪元时间格式的时间戳（以秒为单位）。有效范围在 1-31556889864403199 之间。您也可以指定表达式。

offsetInNanos

（可选）从timeInSeconds转换的纳秒偏移量。有效范围在 0-999999999 之间。您也可以指定表达式。

value

一个包含资产属性值的结构。

> **Important**
>
> 您必须指定以下值类型之一，具体取决于指定的资产属性的 `dataType`。有关更多信息，请参阅《AWS IoT SiteWise API Reference》中的 AssetProperty。

`booleanValue`

（可选）资产属性值是一个布尔值，该值必须为 `TRUE` 或 `FALSE`。您也可以指定表达式。如果使用表达式，则计算结果应为布尔值。

`doubleValue`

（可选）资产属性值为双精度值。您也可以指定表达式。如果使用表达式，则计算结果应为双精度值。

`integerValue`

（可选）资产属性值为整数值。您也可以指定表达式。如果使用表达式，则计算结果应为整数值。

`stringValue`

（可选）资产属性值为字符串。您也可以指定表达式。如果使用表达式，则计算结果应为字符串值。

> **Note**
>
> 确保附加到您的 AWS IoT Events 服务角色的策略授予了该`iotsitewise:BatchPutAssetPropertyValue`权限。有关更多信息，请参阅 的身份和访问管理 AWS IoT Events。

有关更多信息，请参阅《AWS IoT Events API Reference》中的 IotSiteWiseAction。

Amazon DynamoDB

DynamoDB action

Amazon DynamoDB 操作可以让您将数据发送至 DynamoDB 表。DynamoDB 表中有一个列用于接收指定操作有效负载中的所有属性/值对。有关支持的区域列表，请参见Amazon Web Services 一般参考中的 Amazon DynamoDB 端点和配额。

Amazon DynamoDB 是一种全托管 NoSQL 数据库服务，提供快速而可预测的性能，能够实现无缝扩展。有关更多信息，请参阅《Amazon DynamoDB 开发者指南》中的什么是 DynamoDB？。

More information (10)

在向 DynamoDB 表的一列发送数据时，您必须指定以下参数。

`tableName`

接收数据的 DynamoDB 表名称。`tableName` 值必须与 DynamoDB 表的表格名称匹配。您也可以指定表达式。

`hashKeyField`

哈希键（也称为分区键）的名称。`hashKeyField` 值必须与 DynamoDB 表的分区键匹配。您也可以指定表达式。

`hashKeyType`

（可选）哈希键的数据类型。哈希键类型的值必须为 `STRING` 或 `NUMBER`。默认值为 `STRING`。您也可以指定表达式。

`hashKeyValue`

哈希键的值。`hashKeyValue`使用替换模板。这些模板在运行时提供数据。您也可以指定表达式。

`rangeKeyField`

（可选）范围键（也称为排序键）的名称。`rangeKeyField` 值必须与 DynamoDB 表的排序键匹配。您也可以指定表达式。

`rangeKeyType`

（可选）范围键的数据类型。哈希键类型的值必须为 `STRING` 或 `NUMBER`。默认值为 `STRING`。您也可以指定表达式。

`rangeKeyValue`

（可选）范围键的值。`rangeKeyValue`使用替换模板。这些模板在运行时提供数据。您也可以指定表达式。

operation

（可选）要执行的操作类型。您也可以指定表达式。操作值必须是以下值之一：

- `INSERT` - 将数据作为新项插入到 DynamoDB 表中。这是默认值。
- `UPDATE` - 使用新数据更新 DynamoDB 表的现有项。
- `DELETE` - 删除 DynamoDB 表中的现有项。

`payloadField`

（可选）接收操作有效负载的 DynamoDB 列的名称。默认名称为 `payload`。您也可以指定表达式。

`payload`

（可选）默认有效负载包含有关探测器模型实例和触发操作的事件的信息的所有属性/值对。此外，您还可以自定义负载。有关更多信息，请参阅AWS IoT Events 《API 参考》中的有效负载。

如果指定的有效负载类型为字符串，则 `DynamoDBAction` 会将非 JSON 数据以二进制形式写入 DynamoDB 表。DynamoDB 控制台以 Base64 编码文本格式显示数据。`payloadField` 值为 *`payload-field`*`_raw`。您也可以指定表达式。

> **Note**
>
> 确保附加到您的 AWS IoT Events 服务角色的策略授予了该`dynamodb:PutItem`权限。有关更多信息，请参阅 的身份和访问管理 AWS IoT Events。

有关更多信息，请参阅 AWS IoT Events API 参考DBAction中的 Dynamo。

Amazon DynamoDB(v2)

DynamoDBv2 action

The Amazon DynamoDB(v2) 操作可以让您将数据写入 DynamoDB 表。DynamoDB 表中有一个单独的列用于接收指定操作有效负载中的属性/值对。有关支持的区域列表，请参见Amazon Web Services 一般参考中的 Amazon DynamoDB 端点和配额。

Amazon DynamoDB 是一种全托管 NoSQL 数据库服务，提供快速而可预测的性能，能够实现无缝扩展。有关更多信息，请参阅《Amazon DynamoDB 开发者指南》中的什么是 DynamoDB？。

More information (2)

在将数据发送至 DynamoDB 表的多个列时，您必须指定以下参数。

`tableName`

接收数据的 DynamoDB 表名称。您也可以指定表达式。

`payload`

（可选）默认有效负载包含有关探测器模型实例和触发操作的事件的信息的所有属性/值对。此外，您还可以自定义负载。有关更多信息，请参阅AWS IoT Events 《API 参考》中的有效负载。

> **Important**
>
> 有效负载类型必须是 JSON。您也可以指定表达式。

> **Note**
>
> 确保附加到您的 AWS IoT Events 服务角色的策略授予了该`dynamodb:PutItem`权限。有关更多信息，请参阅 的身份和访问管理 AWS IoT Events。

有关更多信息，请参阅 AWS IoT Events API 参考中的 Dynamo DBv2 操作。

Amazon Data Firehose

Firehose action

Amazon Data Firehose 操作允许您将数据发送到 Firehose 传输流。有关支持的区域列表，请参阅中的 Amazon Data Firehose 终端节点和配额。Amazon Web Services 一般参考

Amazon Data Firehose 是一项完全托管的服务，用于向亚马逊简单存储服务（亚马逊简单存储服务）、亚马逊Redshift、亚马逊服务（OpenSearch 服务）和 Splunk 等目的地提供实时流数据。OpenSearch 有关更多信息，请参阅《Amazon Data Firehose 开发人员指南》中的什么是 Amazon Data Firehose？。

More information (3)

向 Firehose 传输流发送数据时，必须指定以下参数。

`deliveryStreamName`

接收数据的 Firehose 传输流的名称。

`separator`

（可选）您可以使用字符分隔符来分隔发送到 Firehose 传输流的连续数据。分隔符值必须是`'\n'`（换行符）、`'\t'`（制表符）、`'\r\n'`（Windows 新行）或`','`（逗号）。

`payload`

（可选）默认有效负载包含有关探测器模型实例和触发操作的事件的信息的所有属性/值对。此外，您还可以自定义负载。有关更多信息，请参阅AWS IoT Events 《API 参考》中的有效负载。

> **Note**
>
> 确保附加到您的 AWS IoT Events 服务角色的策略授予了该`firehose:PutRecord`权限。有关更多信息，请参阅 的身份和访问管理 AWS IoT Events。

有关更多信息，请参阅《AWS IoT Events API Reference》中的 FirehoseAction。

AWS Lambda

Lambda action

该 AWS Lambda 操作允许您调用 Lambda 函数。有关支持的区域列表，请参见 Amazon Web Services 一般参考 中的 AWS Lambda 端点和配额。

AWS Lambda 是一项计算服务，允许您在不预置或管理服务器的情况下运行代码。有关更多信息，请参阅什么是 AWS Lambda？ 在《AWS Lambda 开发人员指南》中。

More information (2)

调用 Lambda 函数时，您必须指定以下参数。

`functionArn`

要调用的 Lambda 函数的 ARN。

`payload`

（可选）默认有效负载包含有关探测器模型实例和触发操作的事件的信息的所有属性/值对。此外，您还可以自定义负载。有关更多信息，请参阅AWS IoT Events 《API 参考》中的有效负载。

> **Note**
>
> 确保附加到您的 AWS IoT Events 服务角色的策略授予了该`lambda:InvokeFunction`权限。有关更多信息，请参阅 的身份和访问管理 AWS IoT Events。

有关更多信息，请参阅《AWS IoT Events API Reference》中的 LambdaAction。

Amazon Simple Notification Service

SNS action

Amazon SNS 主题发布操作可以让您发布 Amazon SNS 消息。有关支持的区域列表，请参阅Amazon Web Services 一般参考中的 Amazon Simple Notification Service 端点和配额。

Amazon Simple Notification Service (Amazon Simple Notification Service) 是一项 Web 服务，用于协调和管理向订阅端点或客户端的消息交付或发送。有关更多信息，请参阅《Amazon Simple Notification Service 开发者指南》中的什么是 Amazon SNS？。

> **Note**
>
> Amazon SNS 主题发布操作不支持Amazon SNS FIFO（先进先出）主题。由于规则引擎是一项完全分布式服务，因此可能无法按 Amazon SNS 操作发起时的指定顺序显示消息。

More information (2)

发布 Amazon SNS 消息时，您必须指定以下参数。

`targetArn`

接收消息的 Amazon SNS 目标的 ARN。

`payload`

（可选）默认有效负载包含有关探测器模型实例和触发操作的事件的信息的所有属性/值对。此外，您还可以自定义负载。有关更多信息，请参阅AWS IoT Events 《API 参考》中的有效负载。

> **Note**
>
> 确保附加到您的 AWS IoT Events 服务角色的策略授予了该`sns:Publish`权限。有关更多信息，请参阅 的身份和访问管理 AWS IoT Events。

有关更多信息，请参阅《AWS IoT Events API Reference》中的 SNSTopicPublishAction。

Amazon Simple Queue Service

SQS action

Amazon SQS 操作可以您将数据发送至 Amazon SQS 队列。有关支持的区域列表，请参阅Amazon Web Services 一般参考中的 Amazon Simple Queue Service 端点和配额。

Amazon Simple Queue Service (Amazon SQS) 提供了一个安全、持久且可用的托管队列，以允许您集成和分离分布式软件系统与组件。有关更多信息，请参阅《Amazon Simple Notification Service 开发者指南》 中的什么是 Amazon Simple Queue Service>。

> **Note**
>
> 亚马逊 SQS 操作不支持 >Amazon SQS FIFO（先入先出）主题。由于规则引擎是一项完全分布式服务，因此可能无法按 Amazon SQ S 操作发起时的指定顺序显示消息。

More information (3)

将数据发送至 Amazon SQS 队列时，您必须指定以下参数。

`queueUrl`

接收数据的 Amazon SQS 队列的 URL。

`useBase64`

（可选）如果您指定，则将数据 AWS IoT Events 编码为 Base64 文本。TRUE默认值为 FALSE。

`payload`

（可选）默认有效负载包含有关探测器模型实例和触发操作的事件的信息的所有属性/值对。此外，您还可以自定义负载。有关更多信息，请参阅AWS IoT Events 《API 参考》中的有效负载。

> **Note**
>
> 确保附加到您的 AWS IoT Events 服务角色的策略授予了该`sqs:SendMessage`权限。有关更多信息，请参阅 的身份和访问管理 AWS IoT Events。

有关更多信息，请参阅《AWS IoT Events API Reference》中的 SNSTopicPublishAction。

您也可以使用 Amazon SNS 和 AWS IoT Core 规则引擎来触发函数。 AWS Lambda 这使得可以使用其他服务（例如 Amazon Connect，甚至是公司的企业资源规划 (ERP) 应用程序）来采取操作。

> **Note**
>
> 要实时收集和处理大量数据记录，您可以使用其他 AWS 服务，例如 Amazon Kinesis。然后，您可以完成初步分析，然后将结果 AWS IoT Events 作为输入发送到探测器。

用于筛选、转换和处理事件数据的表达式

表达式用于评估传入数据、执行计算以及确定应在何种条件下发生特定操作或状态转换。 AWS IoT Events 提供了几种在创建和更新探测器模型时指定值的方法。您可以使用表达式来指定文字值，也 AWS IoT Events 可以在指定特定值之前对表达式求值。

主题

用于筛选设备数据和定义操作的语法 AWS IoT Events

表达式提供了用于筛选设备数据和定义操作的语法。您可以在 AWS IoT Events 表达式中使用文字、运算符、函数、引用和替代模板。通过组合这些组件，您可以创建强大而灵活的表达式来处理物联网数据、执行计算、操作字符串，并在探测器模型中做出合乎逻辑的决策。

文本

- 整数
- 十进制
- 字符串
- 布尔值

运算符

一元运算

- 非运算（布尔）：!
- 非运算（按位）：~
- 减号（算术）：-

字符串

- 联接：+

  两个操作数都必须是字符串。字符串文本必须括在单引号 (') 内。

例如：'my' + 'string' -> 'mystring'

算术

- 加 (**+**)

  两个操作数都必须是数字。

- 减 ：**-**
- 除：**/**

  除法结果为四舍五入的整数值，除非操作数（除数或被除数）中至少有一个是小数值。

- 乘：*****

按位（整数）

- 或：**|**

  例如：13 | 5 -> 13

- 与：**&**

  例如：13 & 5 -> 5

- 异或：**^**

  例如：13 ^ 5 -> 8

- 非：**~**

  例如：~13 -> -14

布尔值

- 小于：**<**
- 小于或等于：**<=**
- 等于：**==**
- 不等于：**!=**
- 大于或等于：**>=**
- 大于：**>**
- 与：**&&**
- 或：**||**

> **Note**
>
> 当 `||` 的子表达式包含未定义的数据时，该子表达式将被视为 `false`。

圆括号

您可以使用圆括号对表达式中的术语进行分组。

要在 AWS IoT Events 表达式中使用的函数

AWS IoT Events 提供了一组内置函数，用于增强探测器模型表达式的功能。这些函数支持计时器管理、类型转换、空值检查、触发器类型识别、输入验证、字符串操作和按位运算。通过利用这些功能，您可以创建响应式 AWS IoT Events 处理逻辑，从而提高物联网应用程序的整体效率。

内置函数

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

如果指定的计时器已过，则计算为 `true`。将 “*`timer-name`*” 替换为您定义的计时器的名称（用引号表示）。在事件操作中，您可以定义计时器，然后启动计时器、重置计时器或清除先前定义的计时器。参见字段 `detectorModelDefinition.states.onInput|onEnter|onExit.events.actions.setTimer.timerName`。

在一种状态下设置的计时器可以在另一种状态下被引用。在进入引用计时器的状态之前，必须访问您创建计时器所处的状态。

例如，探测器模型有两种状态，`TemperatureChecked`和`RecordUpdated`。您创建了一个`TemperatureChecked`处于该状态的计时器。您必须先访问该`TemperatureChecked`州，然后才能在该`RecordUpdated`州使用计时器。

为了确保准确性，计时器应设为最短时间为 60 秒。

> **Note**
>
> `timeout()` 仅在计时器实际到期后首次检查时返回 `true`，之后返回 `false`。

convert(*type*, *expression*)

计算为转换为指定类型的表达式的值。该*type*值必须为String、Boolean、或Decimal。使用其中一个关键字或计算为包含该关键字的字符串的表达式。只有以下转换成功并返回有效值：

- 布尔值 -> 字符串

  返回字符串 "true" 或 "false"。

- 小数值 -> 字符串
- 字符串-> 布尔值
- 字符串-> 小数值

  指定的字符串必须是小数值的有效表示形式，否则 convert() 将失败。

如果 convert() 未返回有效值，则它所属的表达式也无效。此结果等同于 false 且不会触发 actions 或过渡到作为表达式发生的事件的一部分而指定的 nextState。

isNull(*expression*)

在表达式返回为空时计算 true。例如，如果输入 MyInput 收到消息 { "a": null }，则以下计算为 true，但 isUndefined($input.MyInput.a) 计算为 false。

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

isUndefined(*expression*)

如果表达式未定义，则计算为true。例如，如果输入 MyInput 收到消息 { "a": null }，则以下计算为 false，但 isNull($input.MyInput.a) 计算为 true。

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

triggerType("*type*")

type 值可能是 "Message" 或 "Timer"。如果因为计时器已过期而对其出现的事件条件进行评估，则计算为 true，如下例所示。

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

或者收到了输入消息。

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

currentInput("*input*")

如果因为收到了指定的输入消息而对其出现的事件条件进行评估，则计算为 `true`。例如，如果输入 Command 收到消息 `{ "value": "Abort" }`，则以下计算为 `true`。

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

使用此函数来验证是否因为已收到特定输入且计时器尚未过期而正在评估条件，如以下表达式所示。

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

字符串匹配函数

startsWith(*expression1*, *expression2*)

如果第一个字符串表达式以第二个字符串表达式开头，则计算为 `true`。例如，如果输入 MyInput 收到消息 `{ "status": "offline"}`，则以下计算为 `true`。

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

两个表达式的计算结果必须为字符串值。如果任一表达式的计算结果都不是字符串值，则函数的结果未定义。不进行任何转换。

endsWith(*expression1*, *expression2*)

如果第一个字符串表达式以第二个字符串表达式结尾，则计算为 `true`。例如，如果输入 MyInput 收到消息 `{ "status": "offline" }`，则以下计算为 `true`。

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

两个表达式的计算结果必须为字符串值。如果任一表达式的计算结果都不是字符串值，则函数的结果未定义。不进行任何转换。

contains(*expression1*, *expression2*)

如果第一个字符串表达式包含第二个字符串表达式，则计算为 `true`。例如，如果输入 MyInput 收到消息 `{ "status": "offline" }`，则以下计算为 `true`。

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

两个表达式的计算结果必须为字符串值。如果任一表达式的计算结果都不是字符串值，则函数的结果未定义。不进行任何转换。

按位整数操作函数

bitor(*expression1*, *expression2*)

计算整数表达式的按位或（对整数的相应位执行二进制或运算）。例如，如果输入 MyInput 收到消息 { "value1": 13, "value2": 5 }，则以下计算为 13。

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

两个表达式的计算结果都必须为整数值。如果任一表达式的计算结果都不是整数值，则函数的结果未定义。不进行任何转换。

bitand(*expression1*, *expression2*)

计算整数表达式的按位与（对整数的相应位执行二进制与运算）。例如，如果输入 MyInput 收到消息 { "value1": 13, "value2": 5 }，则以下计算为 5。

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

两个表达式的计算结果都必须为整数值。如果任一表达式的计算结果都不是整数值，则函数的结果未定义。不进行任何转换。

bitxor(*expression1*, *expression2*)

计算整数表达式的按位异或（对整数的相应位执行二进制异或运算）。例如，如果输入 MyInput 收到消息 { "value1": 13, "value2": 5 }，则以下计算为 8。

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

两个表达式的计算结果都必须为整数值。如果任一表达式的计算结果都不是整数值，则函数的结果未定义。不进行任何转换。

bitnot(*expression*)

计算整数表达式的按位取反（对整数的位执行二进制取反运算）。例如，如果输入 MyInput 收到消息 { "value": 13 }，则以下计算为 -14。

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

两个表达式的计算结果都必须为整数值。如果任一表达式的计算结果都不是整数值，则函数的结果未定义。不进行任何转换。

AWS IoT Events 表达式中输入和变量的参考

输入

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

input-name是您使用CreateInput操作创建的输入。

例如，如果您有一个名为 TemperatureInput 的输入（您对其定义了 inputDefinition.attributes.jsonPath 条目），则这些值可能会出现在以下可用字段中。

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

要引用该 temperature 字段的值，请使用以下命令。

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

对于值为数组的字段，您可以使用 [*n*] 来引用数组的成员。例如，给定以下值：

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

78.8可以使用以下命令引用该值。

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

变量

$variable.*variable-name*

*variable-name*是您使用CreateDetectorModel操作定义的变量。

例如，如果您使用定义了一个名为 TechnicianID 的变量（您使用 detectorDefinition.states.onInputEvents.actions.setVariable.variableName 定义了该变量），则可以使用以下命令引用最近赋予该变量的（字符串）值。

```
$variable.TechnicianID
```

只能使用 setVariable 操作来设置变量的值。不能为表达式中的变量赋值。变量不能被取消设置。例如，您不能为其分配值 null。

> **Note**
>
> 在使用不遵循（正则表达式）模式[a-zA-Z][a-zA-Z0-9_]*的标识符的引用中，必须用反引号 (`) 将这些标识符括起来。例如，对具有名为 _value 的字段的名为 MyInput 的输入的引用必须将该字段指定为 $input.MyInput.`_value`。

当您在表达式中使用引用时，请检查以下内容：

- 当您将引用值作为一个或多个运算符的操作数时，确保您引用的所有数据类型均兼容。

  例如，在以下表达式中，整数 2 是 == 和 && 运算符的操作数。为确保操作数兼容，$variable.testVariable + 1 和 $variable.testVariable必须引用整数或小数。

  此外，整数 1 是运算符+的操作数。因此，$variable.testVariable 必须引用整数或小数。

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

- 当您使用引用作为传递给函数的自变量时，请确保该函数支持您引用的数据类型。

  例如，以下 timeout("*time-name*") 函数需要带有双引号的字符串作为自变量。如果使用引用作为*timer-name*值，则必须使用双引号引用字符串。

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

> **Note**
>
> 对于convert(*type*, *expression*)函数，如果您使用引用作为*type*值，则引用的计算结果必须是StringDecimal、或Boolean。

AWS IoT Events 表达式支持整数、十进制、字符串和布尔数据类型。下表提供了不兼容的类型对的列表。

不兼容的类型对
整数、字符串
整数、布尔值
小数值、字符串
小数值、布尔值
字符串、布尔值

AWS IoT Events 表达式的替换模板

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

${} 将字符串标识为插值字符串。expression可以是任何 AWS IoT Events 表达式。这包括运算符、函数和引用。

例如，您使用SetVariableAction操作来定义变量。variableName 是 SensorID，value 是 10。您可以创建以下替换模板。

替换模板	结果字符串
`'${'Sensor ' + $variable.SensorID}'`	"Sensor 10"

替换模板	结果字符串
`'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}"`

的表达式示例和用法 AWS IoT Events

您可以通过以下方式指定探测器模型中的值：

- 在 AWS IoT Events 控制台中输入支持的表达式。
- 将表达式 AWS IoT Events APIs 作为参数传递给。

表达式支持文字、运算符、函数、引用和替代模板。

> **Important**
> 您的表达式必须引用整数、小数值、字符串或布尔值。

写 AWS IoT Events 表达式

请参阅以下示例以帮助您编写 AWS IoT Events 表达式：

文本

对于文字值，表达式必须包含单引号。一个布尔值必须是 `true` 或 `false`。

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

参考

对于引用，必须指定变量或输入值。

- 以下输入引用小数值 10.01。

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

- 以下变量引用字符串 Greenhouse Temperature Table。

```
$variable.TableName
```

替换模板

对于替代模板，您必须使用 ${}，且模板必须在单引号内。替代模板还可以包含文字、运算符、函数、引用和替代模板的组合。

- 以下表达式的计算结果是一个字符串 50.018 in Fahrenheit。

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

- 以下表达式的计算结果是一个字符串 {\"sensor_id\":\"Sensor_1\",\"temperature\":\"50.018\"}。

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

字符串连接

对于字符串串联，必须使用 +。字符串串联还可以包含文字、运算符、函数、引用和替代模板的组合。

- 以下表达式的计算结果是一个字符串 Greenhouse Temperature Table 2000-01-01。

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

AWS IoT Events 探测器模型示例

本页提供了演示如何配置各种 AWS IoT Events 功能的示例用例列表。示例范围从温度阈值等基本检测到更高级的异常检测和机器学习场景。每个示例都包含过程和代码片段，可帮助您设置 AWS IoT Events 检测、操作和集成。这些示例展示了该 AWS IoT Events 服务的灵活性，以及如何针对不同的物联网应用和用例对其进行自定义。在探索 AWS IoT Events 功能或需要实施特定检测或自动化工作流程的指导时，请参阅此页面。

主题

示例：将 HVAC 温度控制与 AWS IoT Events

后台故事

此示例实现了具有以下功能的温度控制模型（恒温器）：

- 您定义的一个探测器模型可以监视和控制多个区域。（将为每个区域创建一个探测器实例。）
- 每个探测器实例接收来自放置在每个控制区域的多个传感器的温度数据。
- 您可以随时更改每个区域的所需温度（设定点）。
- 您可以为每个区域定义操作参数并随时更改这些参数。
- 您可以随时向某个区域添加传感器或从中删除传感器。
- 您可以为加热和冷却装置启用最短运行时间，以保护它们免受损坏。
- 探测器将拒绝并报告异常的传感器读数。

- 您可以定义紧急温度设定点。如果任何一个传感器报告的温度高于或低于您定义的设定值，则加热或冷却装置将立即启动，探测器将报告该温度峰值。

该示例演示以下功能：

- 创建事件探测器模型。
- 创建输入。
- 将输入数据采集到探测器模型。
- 评估触发条件。
- 参考条件中的状态变量，并根据条件设置变量的值。
- 参考条件中的计时器，并根据条件设置计时器。
- 采取措施发送 Amazon SNS 和 MQTT 消息。

中暖通空调系统的输入定义 AWS IoT Events

A `seedTemperatureInput`用于为某个区域创建探测器实例并定义其操作参数。

在中 AWS IoT Events 配置暖通空调系统的输入对于有效的气候控制非常重要。此示例说明如何设置用于捕获温度、湿度、占用率和能耗数据等参数的输入。学习定义输入属性、配置数据源和设置预处理规则，以帮助您的探测器模型接收准确、及时的信息，从而实现最佳管理和效率。

使用的 CLI 命令：

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

文件：`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" }
    ]
  }
}
```

响应：

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

如有必要，每个区域的每个传感器都应发送 temperatureInput。

使用的 CLI 命令：

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

文件：temperatureInput.json

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

响应：

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

使用 HVAC 系统的探测器模型定义 AWS IoT Events

areaDetectorModel 定义了每个探测器实例的工作方式。每个 state machine 实例都将摄取温度传感器读数，然后根据这些读数更改状态并发送控制消息。

使用的 CLI 命令：

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

文件：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"
}
```

响应：

```
{
    "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 暖通空调系统的示例 AWS IoT Events

在本示例中，BatchPutMessage 用于为某个区域创建探测器实例并定义初始操作参数。

使用的 CLI 命令：

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

文件：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}"
    }
  ]
}
```

响应：

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

在本示例中，BatchPutMessage 用于报告某个区域中单个传感器的温度传感器读数。

使用的 CLI 命令：

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

文件：temperatureExample.json

```
{
  "messages": [
    {
```

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

响应：

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

在本例中，BatchPutMessage 用于更改某个区域的所需温度。

使用的 CLI 命令：

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

文件：seedSetDesiredTemp.json

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

响应：

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

Area51 探测器实例生成的 Amazon SNS 消息示例：

```
Heating system off command> {
  "eventTime":1557520274729,
  "payload":{
    "actionExecutionId":"f3159081-bac3-38a4-96f7-74af0940d0a4",
    "detector":{
      "detectorModelName":"areaDetectorModel",
      "keyValue":"Area51",
      "detectorModelVersion":"1"
    },
    "eventTriggerDetails":{
      "inputName":"seedTemperatureInput",
      "messageId":"00001",
      "triggerType":"Message"
    },
    "state":{
      "stateName":"start",
      "variables":{
        "sensorCount":10,
        "rangeHigh":30.0,
        "resetMe":false,
        "enteringNewState":true,
        "averageTemperature":20.0,
        "rangeLow":15.0,
        "noDelay":false,
        "allowedError":0.7,
        "desiredTemperature":20.0,
        "anomalousHigh":60.0,
        "reportedTemperature":0.1,
        "anomalousLow":0.0,
        "sensorId":0
      },
      "timers":{}
    }
  },
  "eventName":"resetHeatCool"
}
```

```
Cooling system off command> {
  "eventTime":1557520274729,
  "payload":{
    "actionExecutionId":"98f6a1b5-8f40-3cdb-9256-93afd4d66192",
```

```
    "detector":{
      "detectorModelName":"areaDetectorModel",
      "keyValue":"Area51",
      "detectorModelVersion":"1"
    },
    "eventTriggerDetails":{
      "inputName":"seedTemperatureInput",
      "messageId":"00001",
      "triggerType":"Message"
    },
    "state":{
      "stateName":"start",
      "variables":{
        "sensorCount":10,
        "rangeHigh":30.0,
        "resetMe":false,
        "enteringNewState":true,
        "averageTemperature":20.0,
        "rangeLow":15.0,
        "noDelay":false,
        "allowedError":0.7,
        "desiredTemperature":20.0,
        "anomalousHigh":60.0,
        "reportedTemperature":0.1,
        "anomalousLow":0.0,
        "sensorId":0
      },
      "timers":{}
    }
  },
  "eventName":"resetHeatCool"
}
```

在此示例中，我们使用 DescribeDetector API 来获取有关探测器实例当前状态的信息。

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

响应：

```
{
    "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 以暖通空调系统为例 AWS IoT Events

在本示例中，BatchUpdateDetector 用于更改正在工作的探测器实例的操作参数。

高效的暖通空调系统管理通常需要对多个探测器进行批量更新。本节演示如何使用探测 AWS IoT Events器的批量更新功能。学会同时修改多个控制参数，更新阈值，以便您可以调整设备群中的响应操作，从而提高有效管理大型系统的能力。

使用的 CLI 命令：

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

文件：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": [
        ]
      }
    }
  ]
}
```

响应：

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

AWS IoT Core 规则引擎和 AWS IoT Events

以下规则将 AWS IoT Events MQTT 消息作为影子更新请求消息重新发布。我们假设 AWS IoT Core 为由探测器模型控制的每个区域定义了加热装置和冷却装置。

在此示例中，我们定义了名为 `Area51HeatingUnit` 和 `Area51CoolingUnit` 的元素。

使用的 CLI 命令：

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

文件：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"
        }
      }
    ]
  }
}
```

响应：[空]

使用的 CLI 命令：

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

文件：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"
        }
      }
    ]
  }
}
```

响应：[空]

使用的 CLI 命令：

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

文件：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"
        }
      }
    ]
  }
}
```

响应：[空]

使用的 CLI 命令：

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

文件：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"
        }
      }
    ]
  }
}
```

响应：[空]

示例：一台使用以下方法检测条件的起重机 AWS IoT Events

许多起重机的操作员都希望检测机器何时需要维护或更换，并触发相应的通知。每台起重机都有马达。马达发出包含压力和温度信息的消息（输入）。操作员需要两个级别的事件探测器：

- 起重机级别的事件探测器
- 马达级别的事件探测器

使用来自马达的消息（其中包含 craneId 和 motorid 的元数据），操作员可以使用适当的路由执行两个级别的事件探测器。当满足事件条件时，应向相应的 Amazon SNS 主题发送通知。操作员可以配置探测器模型，这样就不会发出重复的通知。

该示例演示以下功能：

- 创建、读取、更新、删除 (CRUD) 输入。
- 创建、读取、更新、删除 (CRUD) 事件探测器模型和不同版本的事件探测器。
- 将一个输入路由到多个事件探测器。
- 将输入摄入探测器模型。
- 评估触发条件和生命周期事件。
- 能够在条件中引用状态变量并根据条件设置其值。
- 包含定义、状态、触发器评估器和操作执行器的运行时系统编排。
- 使用 SNS 目标在 `ActionsExecutor` 中执行操作。

发送命令以响应中检测到的条件 AWS IoT Events

本页提供了使用 AWS IoT Events 命令设置输入、创建探测器模型和发送模拟传感器数据的示例。这些示例演示了如何利用 AWS IoT Events 来监控电机和起重机等工业设备。

```
#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
```

一种用于起重机监控的 AWS IoT Events 探测器模型

监控您的设备或设备群是否出现故障或运行变化，并在此类事件发生时触发操作。您可以用 JSON 定义检测器模型，用于指定状态、规则和操作。这使您可以监控温度和压力等输入，跟踪阈值违规情况并发送警报。这些示例显示了起重机和电机的探测器模型，用于检测过热问题，并在超过阈值时由 Amazon SNS 发出通知。您可以在不中断监控的情况下更新模型以优化行为。

文件：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"
}
```

更新现有的探测器模型。文件：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"
}
```

文件：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"
}
```

更新现有的探测器模型。文件：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 用于起重机监控的输入

在此示例中，我们演示了如何使用为起重机监控系统设置输入 AWS IoT Events。它捕获压力和温度输入，以说明如何为复杂的工业设备监控构建输入。

文件：pressureInput.json

```
{
    "inputName": "PressureInput",
    "inputDescription": "this is a pressure input description",
    "inputDefinition": {
        "attributes": [
          {"jsonPath": "pressure"}
        ]
    }
}
```

文件：temperatureInput.json

```
{
    "inputName": "TemperatureInput",
    "inputDescription": "this is temperature input description",
    "inputDefinition": {
        "attributes": [
            {"jsonPath": "temperature"}
        ]
    }
}
```

使用发送警报和操作消息 AWS IoT Events

有效的消息处理在起重机监控系统中非常重要。本节介绍如何配置 AWS IoT Events 以处理和响应来自起重机传感器的各种消息类型。根据特定消息设置警报可以帮助您解析、筛选和路由状态更新以触发适当的操作。

文件：highPressureMessage.json

```
{
   "messages": [
        {
           "messageId": "1",
           "inputName": "PressureInput",
           "payload": "{\"craneid\": \"100009\", \"pressure\": 80, \"motorid\":
 \"200009\"}"

        }
    ]
}
```

文件：highTemperatureMessage.json

```
{
   "messages": [
        {
           "messageId": "2",
           "inputName": "TemperatureInput",
           "payload": "{\"craneid\": \"100009\", \"temperature\": 40, \"motorid\":
 \"200009\"}"
        }
    ]
}
```

文件：lowPressureMessage.json

```
{
   "messages": [
        {
           "messageId": "1",
           "inputName": "PressureInput",
           "payload": "{\"craneid\": \"100009\", \"pressure\": 20, \"motorid\":
 \"200009\"}"
```

```
        }
    ]
}
```

文件：lowTemperatureMessage.json

```
{
   "messages": [
        {
           "messageId": "2",
           "inputName": "TemperatureInput",
           "payload": "{\"craneid\": \"100009\", \"temperature\": 20, \"motorid\":
 \"200009\"}"
        }
    ]
}
```

示例：使用传感器和应用程序进行 AWS IoT Events 事件检测

该探测器模型是 AWS IoT Events 控制台提供的模板之一。为方便起见，它包含在此处。

此示例演示 AWS IoT Events了使用传感器数据的应用程序事件检测。它显示了如何创建用于监视指定事件的探测器模型，以便您可以触发适当的操作。您可以创建多个传感器输入、定义复杂的事件条件和设置分级响应机制。

```
{
    "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"
    }
}
```

示例： HeartBeat 用于监控设备连接的设备 AWS IoT Events

该探测器模型是 AWS IoT Events 控制台提供的模板之一。为方便起见，它包含在此处。

心跳缺陷 (DHB) 示例说明了 AWS IoT Events 如何在医疗保健监测中使用。此示例说明如何创建检测器模型来分析心率数据、检测不规则模式并触发适当的响应。学习设置输入、定义阈值和配置潜在心脏问题警报，展示相关医疗保健应用 AWS IoT Events的多功能性。

```
{
    "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"
    }
}
```

示例：中的 ISA 警报 AWS IoT Events

该探测器模型是 AWS IoT Events 控制台提供的模板之一。为方便起见，它包含在此处。

```
{
    "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"
}
```

示例：使用生成一个简单的警报 AWS IoT Events

该探测器模型是 AWS IoT Events 控制台提供的模板之一。为方便起见，它包含在此处。

```
{
    "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"
}
```

使用警报进行监控 AWS IoT Events

AWS IoT Events 警报可帮助您监控数据是否有变化。这些数据可以是您衡量设备和过程的指标。您可以创建警报，从而在突破阈值时发送通知。警报可帮助您检测问题、简化维护并优化设备和进程的性能。

警报是警报模型的实例。警报模型指定了要检测的内容、何时发送通知、谁会收到通知等等。您还可以指定警报状态更改时发生的一个或多个支持的操作。 AWS IoT Events 将从您的数据中派生的输入属性路由到相应的警报。如果您正在监控的数据超出指定范围，则会调用警报。您也可以确认警报或将其设置为暂停模式。

与 AWS IoT SiteWise

您可以使用 AWS IoT Events 警报来监控中的资产属性 AWS IoT SiteWise。 AWS IoT SiteWise 向 AWS IoT Events 警报发送资产属性值。 AWS IoT Events 将警报状态发送到 AWS IoT SiteWise。

AWS IoT SiteWise 还支持外部警报。如果您在外部使用警报， AWS IoT SiteWise 并且有返回警报状态数据的解决方案，则可以选择外部警报。外部警报包含用于摄取警报状态数据的测量属性。

AWS IoT SiteWise 不评估外部警报的状态。此外，当警报状态发生变化时，您无法确认或暂停外部警报。

您可以使用 SiteWise 监控器功能在监控器入口中 SiteWise 查看外部警报的状态。

有关更多信息，请参阅《AWS IoT SiteWise 用户指南》中的 “使用警报监控数据” 和《监控SiteWise 器应用指南》中的 “使用警报监控”。

确认流程

创建警报模型时，您可以选择是否启用确认流。如果您启用确认流，当警报状态发生变化时，您的团队会收到通知。您的团队可以确认警报并留下笔记。例如，您可以列出警报的信息以及为解决问题而要采取的措施。如果您正在监控的数据超出指定范围，则会调用警报。

警报具有以下状态：

`DISABLED`

当警报处于 `DISABLED` 状态时，它还没有准备好评估数据。要启用警报，必须将警报更改为 `NORMAL` 状态。

NORMAL

当警报处于 NORMAL 状态时，它就可以评估数据了。

ACTIVE

如果警报处于 ACTIVE 状态，则会调用警报。您正在监控的数据超出了指定范围。

ACKNOWLEDGED

当警报处于 ACKNOWLEDGED 状态时，警报已被调用并且您确认了警报。

LATCHED

警报已被调用，但一段时间后您仍未确认警报。警报会自动更改为 NORMAL 状态。

SNOOZE_DISABLED

当警报处于 SNOOZE_DISABLED 状态时，警报将在指定的时间段内处于禁用状态。暂停时间过后，警报会自动变为 NORMAL 状态。

在中创建警报模型 AWS IoT Events

您可以使用 AWS IoT Events 警报来监控您的数据，并在突破阈值时收到通知。警报提供用于创建或配置警报模型的参数。您可以使用 AWS IoT Events 控制台或 AWS IoT Events API 来创建或配置警报模型。配置警报模型时，更改会随着新数据的到来而生效。

要求

创建警报模型时应遵循以下要求。

- 您可以创建警报模型来监控中的输入属性 AWS IoT Events 或中的资产属性 AWS IoT SiteWise。
  - 如果您选择在中监控输入属性 AWS IoT Events，则在创建警报模型[在中为模型创建输入 AWS IoT Events](#)之前。
  - 如果您选择监控资产属性，则必须 AWS IoT SiteWise 先在中[创建资产模型](#)，然后才能创建警报模型。
- 您必须拥有允许您的警报执行操作和访问 AWS 资源的 IAM 角色。有关更多信息，请参阅[为 AWS IoT Events设置权限](#)。
- 本教程使用的所有 AWS 资源都必须位于同一个 AWS 区域。

创建警报模型（控制台）

以下内容向您展示了如何创建警报模型以监控 AWS IoT Events 控制台中的 AWS IoT Events 属性。

1. 登录 AWS IoT Events 控制台。
2. 在导航窗格中，选择 警报模型。
3. 在警报模型 页面上，选择 创建警报模型。
4. 在警报模型详细信息部分中，执行以下操作：
   a. 输入唯一名称。
   b. （可选）输入描述。
5. 在 警报目标 部分，执行以下操作：

   > **Important**
   > 如果选择 AWS IoT SiteWise 资产属性，则必须已在 AWS IoT SiteWise中创建了资产模型。

   a. 选择 AWS IoT Events 输入属性。
   b. 选择输入。
   c. 选择输入属性密钥。此输入属性用作创建警报的密钥。 AWS IoT Events 将与此密钥关联的输入路由到警报。

      > **Important**
      > 如果输入消息负载不包含此输入属性密钥，或者该密钥不在密钥中指定的 JSON 路径中，则消息将无法摄取 AWS IoT Events。

6. 在 “阈值定义” 部分，您可以定义用于更改警报状态的 AWS IoT Events 输入属性、阈值和比较运算符。
   a. 在输入属性中，选择要监控的属性。

      每次此输入属性收到新数据时，都会对其进行评估以确定警报的状态。
   b. 对于运算符，选择比较运算符。运算符将您的输入属性与属性的阈值进行比较。

可从以下选项中进行选择：

- > 大于
- >= 大于或等于
- < 小于
- <= 小于或等于
- = 等于
- != 不等于

c. 对于阈值，在输入中 AWS IoT Events 输入一个数字或选择一个属性。 AWS IoT Events 将此值与您选择的输入属性的值进行比较。

d. （可选）对于严重性，请使用您的团队能够理解的数字来反映此警报的严重性。

7. （可选）在通知设置部分，配置警报的通知设置。

您最多可添加 10 个通知。对于通知 1，执行以下操作：

a. 对于 协议，请从以下选项中选择：

- 电子邮件和短信 - 警报会发送短信通知和电子邮件通知。
- 电子邮件 - 警报会发送电子邮件通知。
- 短信 - 警报会发送短信通知。

b. 对于发件人，请指定可以发送有关此警报的通知的电子邮件地址。

要向发件人列表中添加更多电子邮件地址，请选择添加发件人。

c. （可选）在收件人中，选择收件人。

要向收件人列表中添加更多用户，请选择添加新用户。您必须先将新用户添加到您的 IAM Identity Center 存储中，然后才能将其添加到警报模型中。有关更多信息，请参阅 在中管理警报收件人的 IAM 身份中心访问权限 AWS IoT Events。

d. （可选）对于其他自定义消息，请输入一条消息，描述警报检测到的内容以及收件人应采取的操作。

8. 在实例部分，您可以启用或禁用基于此警报模型创建的所有警报实例。

9. 在高级设置部分中，执行以下操作：

a. 对于确认流，您可以启用或禁用通知。

- 如果选择启用，当警报状态发生变化时，您会收到通知。在警报状态可恢复到正常前，您必须选择确认通知。
- 如果选择禁用，则无需执行任何操作。当测量值返回到指定范围以内时，警报会自动变更至正常状态。

有关更多信息，请参阅 确认流程。

b. 对于权限，请选择下列选项之一：

- 您可以通过 AWS 策略模板创建新角色并 AWS IoT Events 自动为您创建 IAM 角色。
- 您可以使用允许此警报模型执行操作和访问其他 AWS 资源的现有 IAM 角色。

有关更多信息，请参阅 AWS IoT Events的身份和访问权限管理。

c. 对于其他通知设置，您可以编辑您的 AWS Lambda 功能以管理警报通知。为您的 AWS Lambda 函数选择以下选项之一：

- 创建新 AWS Lambda 函数- AWS IoT Events 为您创建新 AWS Lambda 函数。
- 使用现有 AWS Lambda 函数-通过选择 AWS Lambda 函数名称来使用现有 AWS Lambda 函数。

有关可能操作的更多信息，请参阅 AWS IoT Events 使用其他 AWS 服务。

d. （可选）在 “设置状态操作” 中，您可以添加一个或多个在警报状态发生变化时要 AWS IoT Events 执行的操作。

10. (可选) 您可以添加标签来管理警报。有关更多信息，请参阅标记您的 AWS IoT Events 资源。
11. 选择创建。

对警报做出响应 AWS IoT Events

有效响应警报是管理物联网系统的一个重要方面 AWS IoT Events。探索配置和处理警报的各种方法，包括：设置通知渠道、定义上报程序和实施自动响应操作。学习如何创建细致入微的警报条件，确定警报的优先级，并与其他 AWS 服务集成，为您的物联网应用构建响应式警报管理系统。

如果启用确认流 ，当警报状态发生变化时，您会收到通知。要响应警报，您可以确认、禁用、启用、重置或暂停警报。

Console

以下内容向您展示了如何在 AWS IoT Events 控制台中响应警报。

1. 登录 AWS IoT Events 控制台。
2. 在导航窗格中，选择 警报模型。
3. 选择目标警报模型。
4. 在警报列表部分，选择目标警报。
5. 您可以从操作中选择以下选项之一：

   - 确认 - 警报变为 ACKNOWLEDGED 状态。
   - 禁用 - 警报变为 DISABLED 状态。
   - 启用 - 警报变为 NORMAL 状态。
   - 重置 - 警报变为 NORMAL 状态。
   - 暂停，然后执行以下操作：

     1. 选择 暂停时长或输入自定义暂停时长。
     2. 选择保存。

     警报变为 SNOOZE_DISABLED 状态

   有关警报状态的更多信息，请参阅 确认流程。

API

要响应一个或多个警报，您可以使用以下 AWS IoT Events API 操作：

- BatchAcknowledgeAlarm
- BatchDisableAlarm
- BatchEnableAlarm
- BatchResetAlarm
- BatchSnoozeAlarm

在中管理警报通知 AWS IoT Events

AWS IoT Events 与 Lambda 集成，提供自定义事件处理功能。本节探讨如何在探 AWS IoT Events 测器模型中使用 Lambda 函数，使您能够执行复杂逻辑、与外部服务交互以及实现复杂的事件处理。

AWS IoT Events 使用 Lambda 函数来管理警报通知。您可以使用提供的 Lambda 函数， AWS IoT Events 也可以创建一个新的函数。

主题

在中创建 Lambda 函数 AWS IoT Events

AWS IoT Events 提供了 Lambda 函数，使警报能够发送和接收电子邮件和短信通知。

要求

为警报创建 Lambda 函数时应遵循以下要求：

- 如果您的警报发送短信通知，请确保将 Amazon SNS 配置为发送短信。
  - 有关更多信息，请参阅以下文档：
    - 亚马逊简单通知服务开发者指南中使用亚马逊 SNS 和 O rigination 身份发送亚马逊 SNS 短信的移动短信。
    - 什么是 AWS 最终用户消息短信？ 在《AWS SMS 用户指南》中。
- 如果您的警报发送电子邮件或短信通知，则您必须具有允许 AWS Lambda 使用 Amazon SES 和 Amazon SNS 的 IAM 角色。

  示例策略：

  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" : "*"
        }
    ]
}
```

- 必须为 AWS IoT Events 和选择相同的 AWS 区域 AWS Lambda。有关支持的区域列表，请参阅 Amazon Web Services 一般参考 中的AWS IoT Events 端点和配额以及AWS Lambda 端点和配额。

部署 Lambda 函数以供使用 AWS IoT Events CloudFormation

本教程使用 CloudFormation 模板部署 Lambda 函数。此模板会自动创建一个 IAM 角色，该角色允许 Lambda 函数与 Amazon SES 和亚马逊 SNS 配合使用。

以下内容向您展示了如何使用 AWS Command Line Interface (AWS CLI) 创建 CloudFormation 堆栈。

1. 在设备的终端中，运行`aws --version`以检查您是否安装了 AWS CLI。有关更多信息，请参阅《AWS Command Line Interface 用户指南》中的安装或更新到最新版本的 AWS CLI。
2. 运行`aws configure list`以检查您是否 AWS CLI 在包含本教程所有 AWS 资源的 AWS 区域中配置了。有关更多信息，请参阅《AWS Command Line Interface 用户指南》中的使用命令设置和查看配置设置
3. 下载 CloudFormation 模板 notificationLambda.templat e.yaml.zip。

   > **Note**
   >
   > 如果您在下载文件时遇到困难，也可以在 CloudFormation 模板 中找到该模板。

4. 解压缩内容并将其作为 `notificationLambda.template.yaml` 保存在本地。
5. 在您的设备上打开终端，导航到下载了 `notificationLambda.template.yaml` 文件的目录。
6. 要创建 CloudFormation 堆栈，请运行以下命令：

   ```
   aws cloudformation create-stack --stack-name notificationLambda-stack --template-
   body file://notificationLambda.template.yaml --capabilities CAPABILITY_IAM
   ```

您可以修改此 CloudFormation 模板以自定义 Lambda 函数及其行为。

> **Note**
>
> AWS Lambda 重试函数错误两次。如果该函数没有足够的容量来处理所有传入请求，则事件可能会在队列中等待数小时或数天才能发送到该函数。您可以在函数上配置未送达消息队列 (DLQ) 以捕获未成功处理的事件。有关更多信息，请参阅 AWS Lambda 开发人员指南中的异步调用。

您也可以在 CloudFormation 控制台中创建或配置堆栈。有关更多信息，请参阅 AWS CloudFormation 用户指南中的使用堆栈。

为创建自定义 Lambda 函数 AWS IoT Events

您可以创建 Lambda 函数或修改 AWS IoT Events提供的函数。

创建自定义 Lambda 函数时应遵循以下要求。

- 添加允许您的 Lambda 函数执行指定操作和访问 AWS 资源的权限。
- 如果您使用提供的 Lambda 函数 AWS IoT Events，请务必选择 Python 3.7 运行时。

Lambda 函数示例：

```
import boto3
import json
import logging
import datetime
logger = logging.getLogger()
logger.setLevel(logging.INFO)
ses = boto3.client('ses')
sns = boto3.client('sns')
def check_value(target):
  if target:
    return True
  return False

# Check whether email is verified. Only verified emails are allowed to send emails to
 or from.
def check_email(email):
  if not check_value(email):
    return False
  result = ses.get_identity_verification_attributes(Identities=[email])
  attr = result['VerificationAttributes']
  if (email not in attr or attr[email]['VerificationStatus'] != 'Success'):
      logging.info('Verification email for {} sent. You must have all the emails
 verified before sending email.'.format(email))
      ses.verify_email_identity(EmailAddress=email)
      return False
  return True

# Check whether the phone holder has opted out of receiving SMS messages from your
 account
def check_phone_number(phone_number):
  try:
    result = sns.check_if_phone_number_is_opted_out(phoneNumber=phone_number)
    if (result['isOptedOut']):
        logger.info('phoneNumber {} is not opt in of receiving SMS messages. Phone
 number must be opt in first.'.format(phone_number))
        return False
    return True
```

```
  except Exception as e:
    logging.error('Your phone number {} must be in E.164 format in SSO. Exception
 thrown: {}'.format(phone_number, e))
    return False

def check_emails(emails):
  result = True
  for email in emails:
      if not check_email(email):
          result = False
  return result

def lambda_handler(event, context):
  logging.info('Received event: ' + json.dumps(event))
  nep = json.loads(event.get('notificationEventPayload'))
  alarm_state = nep['alarmState']
  default_msg = 'Alarm ' + alarm_state['stateName'] + '\n'
  timestamp =
 datetime.datetime.utcfromtimestamp(float(nep['stateUpdateTime'])/1000).strftime('%Y-
%m-%d %H:%M:%S')
  alarm_msg = "{} {} {} at {} UTC ".format(nep['alarmModelName'], nep.get('keyValue',
 'Singleton'), alarm_state['stateName'], timestamp)
  default_msg += 'Sev: ' + str(nep['severity']) + '\n'
  if (alarm_state['ruleEvaluation']):
    property = alarm_state['ruleEvaluation']['simpleRule']['inputProperty']
    default_msg += 'Current Value: ' + str(property) + '\n'
    operator = alarm_state['ruleEvaluation']['simpleRule']['operator']
    threshold = alarm_state['ruleEvaluation']['simpleRule']['threshold']
    alarm_msg += '({} {} {})'.format(str(property), operator, str(threshold))
  default_msg += alarm_msg + '\n'

  emails = event.get('emailConfigurations', [])
  logger.info('Start Sending Emails')
  for email in emails:
    from_adr = email.get('from')
    to_adrs = email.get('to', [])
    cc_adrs = email.get('cc', [])
    bcc_adrs = email.get('bcc', [])
    msg = default_msg + '\n' + email.get('additionalMessage', '')
    subject = email.get('subject', alarm_msg)
    fa_ver = check_email(from_adr)
    tas_ver = check_emails(to_adrs)
    ccas_ver = check_emails(cc_adrs)
    bccas_ver = check_emails(bcc_adrs)
```

```
    if (fa_ver and tas_ver and ccas_ver and bccas_ver):
      ses.send_email(Source=from_adr,
                     Destination={'ToAddresses': to_adrs, 'CcAddresses': cc_adrs,
'BccAddresses': bcc_adrs},
                     Message={'Subject': {'Data': subject}, 'Body': {'Text': {'Data':
msg}}})
      logger.info('Emails have been sent')

 logger.info('Start Sending SNS message to SMS')
 sns_configs = event.get('smsConfigurations', [])
 for sns_config in sns_configs:
   sns_msg = default_msg + '\n' + sns_config.get('additionalMessage', '')
   phone_numbers = sns_config.get('phoneNumbers', [])
   sender_id = sns_config.get('senderId')
   for phone_number in phone_numbers:
       if check_phone_number(phone_number):
         if check_value(sender_id):
           sns.publish(PhoneNumber=phone_number, Message=sns_msg,
MessageAttributes={'AWS.SNS.SMS.SenderID':{'DataType': 'String','StringValue':
sender_id}})
         else:
           sns.publish(PhoneNumber=phone_number, Message=sns_msg)
         logger.info('SNS messages have been sent')
```

有关更多信息，请参阅 AWS Lambda 开发人员指南中的什么是 AWS Lambda？

CloudFormation 模板

使用以下 CloudFormation 模板创建您的 Lambda 函数。

```
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')
```

使用由提供的 Lambda 函数 AWS IoT Events

AWS IoT Events 对于警报通知，您可以使用提供的 Lambda 函数来管理警报通知。

当您使用 AWS IoT Events 提供的 Lambda 函数来管理警报通知时，应遵循以下要求：

- 您必须验证在 Amazon Simple Email Service (Amazon SES) 中发送电子邮件通知的电子邮件地址。有关更多信息，请参阅《Amazon Simple Email Service 开发人员指南》中的验证电子邮件地址身份。

  如果您收到验证链接，请单击该链接以验证您的电子邮件地址。您也可以查看垃圾邮件文件夹中是否有验证邮件。
- 如果您的警报发送短信通知，则必须使用 E.164 国际电话号码格式作为电话号码。此格式包含 `+<country-calling-code><area-code><phone-number>`。

  电话号码示例：

国家/地区	本地电话号码	E.164 格式的数字
美国	206-555-0100	+12065550100
英国	020-1234-1234	+442012341234
立陶宛	8+601+12345	+37060112345

  要查找国家/地区呼叫码，请访问 countrycode.org。

  提供的 Lambda 函数 AWS IoT Events 会检查您是否使用 E.164 格式的电话号码。但是，它不验证电话号码。如果您确保输入了准确的电话号码，但没有收到短信通知，则可以联系电话运营商。运营商可能会屏蔽消息。

在中管理警报收件人的 IAM 身份中心访问权限 AWS IoT Events

AWS IoT Events AWS IAM Identity Center 用于管理警报接收者的 SSO 访问权限。为 AWS IoT Events 通知收件人实施 IAM 身份中心可以增强安全性和用户体验。要使警报能够向收件人发送通知，您必须启用 IAM Identity Center 并将收件人添加到您的 IAM Identity Center 存储。有关更多信息，请参阅《AWS IAM Identity Center 用户指南》中的添加用户。

Important

- 您必须为 AWS IoT Events AWS Lambda、和 IAM 身份中心选择相同的 AWS 区域。
- AWS Organizations 一次仅支持一个 IAM 身份中心区域。如果您想在其他区域提供 IAM Identity Center，则必须先删除当前的 IAM Identity Center 配置。有关更多信息，请参阅《AWS IAM Identity Center 用户指南》中的IAM Identity Center 区域数据。

安全性 AWS IoT Events

云安全 AWS 是重中之重。作为 AWS 客户，您可以受益于专为满足大多数安全敏感型组织的要求而构建的数据中心和网络架构。

安全是双方 AWS 的共同责任。责任共担模式将其描述为云的 安全性和云中 的安全性：

- 云安全 — AWS 负责保护在 AWS 云中运行 AWS 服务的基础架构。 AWS 还为您提供可以安全使用的服务。作为 AWS 合规性计划的一部分，我们的安全措施的有效性定期由第三方审计员进行测试和验证。要了解适用的合规计划 AWS IoT Events，请参阅按合规计划划分的范围内的AWS 服务。
- 云端安全-您的责任由您使用的 AWS 服务决定。您还需要对其他因素负责，包括您的数据的敏感性、您组织的要求以及适用的法律法规。

本文档将帮助您了解在使用时如何应用分担责任模型 AWS IoT Events。以下主题向您介绍如何进行配置 AWS IoT Events 以满足您的安全和合规性目标。您还将学习如何使用其他 AWS 服务来帮助您监控和保护您的 AWS IoT Events 资源。

主题

的身份和访问管理 AWS IoT Events

AWS Identity and Access Management (IAM) 是一项 AWS 服务，可帮助管理员安全地控制对 AWS 资源的访问。IAM 管理员控制谁可以进行身份验证（登录）和授权（拥有权限）使用 AWS IoT Events 资源。IAM 是一项无需额外付费即可使用的 AWS 服务。

主题

- 如何 AWS IoT Events 与 IAM 配合使用
- AWS IoT Events 基于身份的策略示例
- 跨服务混淆了副手的预防 AWS IoT Events
- 对 AWS IoT Events 身份和访问进行故障排除

受众

您的使用方式 AWS Identity and Access Management (IAM) 因您的角色而异：

- 服务用户：如果您无法访问功能，请从管理员处请求权限（请参阅对 AWS IoT Events 身份和访问进行故障排除）
- 服务管理员：确定用户访问权限并提交权限请求（请参阅如何 AWS IoT Events 与 IAM 配合使用）
- IAM 管理员：编写用于管理访问权限的策略（请参阅AWS IoT Events 基于身份的策略示例）

使用身份进行身份验证

身份验证是您 AWS 使用身份凭证登录的方式。您必须以 IAM 用户身份进行身份验证 AWS 账户根用户，或者通过担任 IAM 角色进行身份验证。

您可以使用来自身份源的证书 AWS IAM Identity Center（例如（IAM Identity Center）、单点登录身份验证或 Google/Facebook 证书，以联合身份登录。有关登录的更多信息，请参阅《AWS 登录 用户指南》中的如何登录您的 AWS 账户。

对于编程访问，AWS 提供 SDK 和 CLI 来对请求进行加密签名。有关更多信息，请参阅《IAM 用户指南》中的适用于 API 请求的AWS 签名版本 4。

AWS 账户 root 用户

创建时 AWS 账户，首先会有一个名为 AWS 账户 root 用户的登录身份，该身份可以完全访问所有资源 AWS 服务 和资源。我们强烈建议不要使用根用户进行日常任务。有关要求根用户凭证的任务，请参阅《IAM 用户指南》中的需要根用户凭证的任务。

IAM 用户和群组

IAM 用户是对某个人员或应用程序具有特定权限的一个身份。建议使用临时凭证，而非具有长期凭证的 IAM 用户。有关更多信息，请参阅 IAM 用户指南中的要求人类用户使用身份提供商的联合身份验证才能 AWS 使用临时证书进行访问。

IAM 组指定一组 IAM 用户，便于更轻松地对大量用户进行权限管理。有关更多信息，请参阅《IAM 用户指南》中的 IAM 用户使用案例。

IAM 角色

IAM 角色是具有特定权限的身份，可提供临时凭证。您可以通过从用户切换到 IAM 角色（控制台）或调用 AWS CLI 或 AWS API 操作来代入角色。有关更多信息，请参阅《IAM 用户指南》中的担任角色的方法。

IAM 角色对于联合用户访问、临时 IAM 用户权限、跨账户访问、跨服务访问以及在 Amazon EC2 上运行的应用程序非常有用。有关更多信息，请参阅《IAM 用户指南》中的 IAM 中的跨账户资源访问。

使用策略管理访问

您可以 AWS 通过创建策略并将其附加到 AWS 身份或资源来控制中的访问权限。策略定义了与身份或资源关联时的权限。 AWS 在委托人提出请求时评估这些政策。大多数策略都以 JSON 文档的 AWS 形式存储在中。有关 JSON 策略文档的更多信息，请参阅《IAM 用户指南》中的 JSON 策略概述。

管理员使用策略，通过定义哪个主体可以在什么条件下对哪些资源执行哪些操作来指定谁有权访问什么。

默认情况下，用户和角色没有权限。IAM 管理员创建 IAM 策略并将其添加到角色中，然后用户可以担任这些角色。IAM 策略定义权限，与执行操作所用的方法无关。

基于身份的策略

基于身份的策略是您附加到身份（用户、组或角色）的 JSON 权限策略文档。这些策略控制身份可以执行什么操作、对哪些资源执行以及在什么条件下执行。要了解如何创建基于身份的策略，请参阅《IAM 用户指南》中的使用客户管理型策略定义自定义 IAM 权限。

基于身份的策略可以是内联策略（直接嵌入到单个身份中）或托管策略（附加到多个身份的独立策略）。要了解如何在托管策略和内联策略之间进行选择，请参阅《IAM 用户指南》中的在托管策略与内联策略之间进行选择。

其他策略类型

AWS 支持其他策略类型，这些策略类型可以设置更常见的策略类型授予的最大权限：

- 权限边界 – 设置基于身份的策略可以授予 IAM 实体的最大权限。有关更多信息，请参阅《 IAM 用户指南》中的 IAM 实体的权限边界。

- 服务控制策略 (SCPs)-在中指定组织或组织单位的最大权限 AWS Organizations。有关更多信息，请参阅《AWS Organizations 用户指南》中的服务控制策略。
- 资源控制策略 (RCPs)-设置账户中资源的最大可用权限。有关更多信息，请参阅《AWS Organizations 用户指南》中的资源控制策略 (RCPs)。
- 会话策略 – 在为角色或联合用户创建临时会话时，作为参数传递的高级策略。有关更多信息，请参阅《IAM 用户指南》中的会话策略。

多个策略类型

当多个类型的策略应用于一个请求时，生成的权限更加复杂和难以理解。要了解在涉及多种策略类型时如何 AWS 确定是否允许请求，请参阅 IAM 用户指南中的策略评估逻辑。

有关身份和访问管理的更多信息

有关身份和访问管理的更多信息 AWS IoT Events，请继续访问以下页面：

- 如何 AWS IoT Events 与 IAM 配合使用
- 对 AWS IoT Events 身份和访问进行故障排除

如何 AWS IoT Events 与 IAM 配合使用

在使用 IAM 管理访问权限之前 AWS IoT Events，您应该了解哪些可用的 IAM 功能 AWS IoT Events。要全面了解如何 AWS IoT Events 和其他 AWS 服务与 IAM 配合使用，请参阅 IAM 用户指南中的与 IAM 配合使用的AWS 服务。

主题

- AWS IoT Events 基于身份的策略
- AWS IoT Events 基于资源的政策
- 基于 AWS IoT Events 标签的授权
- AWS IoT Events IAM 角色

AWS IoT Events 基于身份的策略

使用 IAM 基于身份的策略，您可以指定允许或拒绝的操作和资源，以及指定在什么条件下允许或拒绝操作。 AWS IoT Events 支持特定操作、资源和条件键。要了解在 JSON 策略中使用的所有元素，请参阅《IAM 用户指南》中的 IAM JSON 策略元素参考。

操作

基于 IAM 身份的策略的 Action 元素描述该策略将允许或拒绝的特定操作。策略操作通常与关联的 AWS API 操作同名。此策略用于策略中以授予执行关联操作的权限。

正在执行的策略操作在操作前 AWS IoT Events 使用以下前缀:iotevents:. 例如，要授予某人使用 AWS IoT Events CreateInput API 操作创建 AWS IoT Events 输入的权限，您需要将该iotevents:CreateInput操作包含在他们的策略中。要授予某人 AWS IoT Events BatchPutMessage通过 API 操作发送输入的权限，您需要将该iotevents-data:BatchPutMessage操作包含在他们的策略中。策略声明必须包含Action或NotAction元素。AWS IoT Events 定义了它自己的一组操作，这些操作描述了您可以使用此服务执行的任务。

要在单个语句中指定多项操作，请使用逗号将它们隔开，如下所示：

```
"Action": [
      "iotevents:action1",
      "iotevents:action2"
```

您也可以使用通配符 （*) 指定多个操作。例如，要指定以单词 Describe 开头的所有操作，包括以下操作：

```
"Action": "iotevents:Describe*"
```

要查看 AWS IoT Events 操作列表，请参阅 IAM 用户指南 AWS IoT Events中的定义操作。

资源

Resource 元素指定要向其应用操作的对象。语句必须包含 Resource 或 NotResource 元素。您可使用 ARN 来指定资源，或使用通配符 (*) 以指明该语句适用于所有资源。

探 AWS IoT Events 测器模型资源具有以下 ARN：

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

有关格式的更多信息 ARNs，请参阅使用 Amazon 资源名称识别 AWS 资源 (ARNs)。

例如，要在语句中指定 Foobar 探测器模型，请使用以下 ARN：

```
"Resource": "arn:aws:iotevents:us-east-1:123456789012:detectorModel/Foobar"
```

要指定属于特定账户的所有实例，请使用通配符 (*)：

```
"Resource": "arn:aws:iotevents:us-east-1:123456789012:detectorModel/*"
```

某些 AWS IoT Events 操作（例如创建资源的操作）无法对特定资源执行。在这些情况下，您必须使用通配符（*)。

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

某些 AWS IoT Events API 操作涉及多个资源。例如，`CreateDetectorModel` 在其条件语句中引用输入，所以用户必须获得相应权限才能使用该输入和探测器模型。要在单个语句中指定多个资源，请 ARNs 用逗号分隔。

```
"Resource": [
      "resource1",
      "resource2"
```

要查看 AWS IoT Events 资源类型及其列表 ARNs，请参阅 IAM 用户指南 AWS IoT Events中的由定义的资源。要了解您可以在哪些操作中指定每个资源的 ARN，请参阅 AWS IoT Events定义的操作。

条件键

在 `Condition` 元素（或 `Condition` 块）中，可以指定语句生效的条件。`Condition` 元素是可选的。您可以构建使用条件运算符（例如，等于或小于）的条件表达式，以使策略中的条件与请求中的值相匹配。

如果您在一个语句中指定多个 `Condition` 元素，或在单个 `Condition` 元素中指定多个键，则 AWS 使用逻辑 AND 运算评估它们。如果您为单个条件键指定多个值，则使用逻辑OR运算来 AWS 评估条件。在授予语句的权限之前必须满足所有的条件。

在指定条件时，您也可以使用占位符变量。例如，仅当用户使用其用户名进行标记时，您才可为其授予访问资源的权限。有关更多信息，请参阅 IAM 用户指南 中的 IAM 策略元素：变量和标签。

AWS IoT Events 不提供任何特定于服务的条件密钥，但它确实支持使用某些全局条件密钥。要查看所有 AWS 全局条件键，请参阅 IAM 用户指南中的AWS 全局条件上下文密钥。”

示例

要查看 AWS IoT Events 基于身份的策略的示例，请参阅。AWS IoT Events 基于身份的策略示例

AWS IoT Events 基于资源的政策

AWS IoT Events 不支持基于资源的策略。” 要查看详细的基于资源的策略页面示例，请参阅 https://docs.aws.amazon.com/lambda/latest/dg/access-control-resource-based.html。

基于 AWS IoT Events 标签的授权

您可以为 AWS IoT Events 资源附加标签或在请求中传递标签 AWS IoT Events。要基于标签控制访问，您需要使用 `iotevents:ResourceTag/key-name`aws:RequestTag/*key-name* 或 `aws:TagKeys` 条件键在策略的条件元素中提供标签信息。有关标记 AWS IoT Events 资源的更多信息，请参阅 标记您的资源 AWS IoT Events

要查看基于身份的策略（用于根据资源上的标签来限制对该资源的访问）的示例，请参阅 根据标签查看 AWS IoT Events 输入。

AWS IoT Events IAM 角色

I AM 角色是您内部具有特定权限 AWS 账户 的实体。

将临时证书与 AWS IoT Events

可以使用临时凭证进行联合身份验证登录，分派 IAM 角色或分派跨账户角色。您可以通过调用 AWS Security Token Service (AWS STS) API 操作（例如AssumeRole或）来获取临时安全证书GetFederationToken。

AWS IoT Events 不支持使用临时证书。

服务关联角色

服务相关角色允许 AWS 服务访问其他服务中的资源以代表您完成操作。服务关联角色显示在 IAM 账户中，并归该服务所有。IAM 管理员可以查看但不能编辑服务关联角色的权限。

AWS IoT Events 不支持服务相关角色。

服务角色

此功能允许服务代表您担任服务角色。此角色允许服务访问其他服务中的资源以代表您完成操作。服务角色显示在 IAM 账户中，并归该账户所有。这意味着，IAM 管理员可以更改该角色的权限。但是，这样做可能会中断服务的功能。

AWS IoT Events 支持服务角色。

AWS IoT Events 基于身份的策略示例

默认情况下，用户和角色无权创建或修改 AWS IoT Events 资源。他们也无法使用 AWS 管理控制台 AWS CLI、或 AWS API 执行任务。IAM 管理员必须创建 IAM 策略，以便为用户和角色授予权限以对所需的指定资源执行特定的 API 操作。然后，管理员必须将这些策略附加到需要这些权限的用户或组。

要了解如何使用这些示例 JSON 策略文档创建 IAM 基于身份的策略，请参阅《IAM 用户指南》中的在 JSON 选项卡上创建策略。

主题

- 策略最佳实践
- 使用控制 AWS IoT Events 台
- 允许用户在中查看自己的权限 AWS IoT Events
- 访问一个 AWS IoT Events 输入
- 根据标签查看 AWS IoT Events 输入

策略最佳实践

基于身份的策略非常强大。它们决定是否有人可以在您的账户中创建、访问或删除 AWS IoT Events 资源。这些操作可能会使 AWS 账户产生成本。创建或编辑基于身份的策略时，请遵循以下指南和建议：

- 开始使用 AWS 托管策略 — 要 AWS IoT Events 快速开始使用，请使用 AWS 托管策略为员工提供所需的权限。这些策略已在您的账户中提供，并由 AWS维护和更新。有关更多信息，请参阅 IAM 用户指南中的使用 AWS 托管策略的权限入门。
- 授予最低权限：创建自定义策略时，仅授予执行任务所需的许可。最开始只授予最低权限，然后根据需要授予其它权限。这样做比起一开始就授予过于宽松的权限而后再尝试收紧权限来说更为安全。有关更多信息，请参阅 IAM 用户指南 中的授予最低权限。
- 为敏感操作启用 MFA – 为增强安全性，要求用户使用多重身份验证 (MFA) 来访问敏感资源或 API 操作。要了解更多信息，请参阅 IAM 用户指南中的在 AWS中使用多重身份验证（MFA)。
- 使用策略条件来增强安全性：在切实可行的范围内，定义基于身份的策略在哪些情况下允许访问资源。例如，您可编写条件来指定请求必须来自允许的 IP 地址范围。您也可以编写条件，以便仅允许指定日期或时间范围内的请求，或者要求使用 SSL 或 MFA。有关更多信息，请参阅 IAM 用户指南中的 IAM JSON 策略元素：条件。

使用控制 AWS IoT Events 台

要访问 AWS IoT Events 控制台，您必须拥有一组最低权限。这些权限必须允许您列出和查看有关您的 AWS IoT Events 资源的详细信息 AWS 账户。如果创建比必需的最低权限更为严格的基于身份的策略，对于附加了该策略的实体（用户或角色），控制台将无法按预期正常运行。

为确保这些实体仍然可以使用 AWS IoT Events 控制台，还要将以下 AWS 托管策略附加到这些实体。有关更多信息，请参阅 IAM 用户指南中的为用户添加权限：

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"
```

```
        }
    ]
}
```

对于仅调用 AWS CLI 或 AWS API 的用户，您无需为其设置最低控制台权限。相反，只允许访问与您尝试执行的 API 操作相匹配的操作。

允许用户在中查看自己的权限 AWS IoT Events

此示例显示您可以如何创建策略，以便允许 用户查看附加到其用户身份的内联和托管策略。允许用户查看自己的 IAM 权限对于提高安全意识和自助服务功能非常有用。此策略包括在控制台上或使用 AWS CLI 或 AWS API 以编程方式完成此操作的权限。

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": "*"
        }
    ]
}
```

访问一个 AWS IoT Events 输入

对 AWS IoT Events 输入进行精细访问控制对于维护多用户或多团队环境中的安全性非常重要。本节介绍如何创建 IAM 策略来授予对特定 AWS IoT Events 输入的访问权限，同时限制其他输入的访问权限。

在此示例中，您可以授予用户 AWS 账户 访问您的其中一个 AWS IoT Events 输入的权限`exampleInput`。您还可以允许用户添加、更新和删除输入。

该策略为用户授予 `iotevents:ListInputs`、`iotevents:DescribeInput`、`iotevents:CreateInput`、`iotevents:D` 和 `iotevents:UpdateInput` 权限。有关向用户授予权限并使用控制台对其进行测试的亚马逊简单存储服务 (Amazon S3) 的示例演练，请参阅使用用户策略控制对存储桶的访问权限。

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/*"
        }
    ]
}
```

根据标签查看 AWS IoT Events 输入

标签可帮助您整理 AWS IoT Events 资源。您可以使用基于身份的策略中的条件根据标签控制对 AWS IoT Events 资源的访问权限。此示例说明如何创建允许查看的策略*input*。但是，仅当 *input* 标签 Owner 具有该用户的用户名的值时，才授予此权限。此策略还授予在控制台上完成此操作的必要权限。

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}"}
            }
        }
    ]
}
```

您可以将此策略附加到您账户中的用户。如果名为的用户`richard-roe`尝试查看 AWS IoT Events *`input`*，则*`input`*必须将其标记为`Owner=richard-roe`或`owner=richard-roe`。否则，将拒绝其访问。条件标签键 `Owner` 匹配 `Owner` 和 `owner`，因为条件键名称不区分大小写。有关更多信息，请参阅《IAM 用户指南》中的 IAM JSON 策略元素：条件。

跨服务混淆了副手的预防 AWS IoT Events

> **Note**
>
> - 该 AWS IoT Events 服务仅允许您使用角色在创建资源的同一个账户中启动操作。这有助于防止出现混乱的副手攻击 AWS IoT Events。
> - 此页面可作为参考，以了解混乱的副手问题是如何运作的，如果 AWS IoT Events 服务中允许使用跨账户资源，则可以避免。

混淆代理问题是一个安全性问题，即不具有某操作执行权限的实体可能会迫使具有更高权限的实体执行该操作。在中 AWS，跨服务模仿可能会导致混乱的副手问题。

一个服务（呼叫服务）调用另一项服务（所谓的服务）时，可能会发生跨服务模拟。可以操纵调用服务以使用其权限对另一个客户的资源进行操作，否则该服务不应有访问权限。为了防止这种情况，我们 AWS 提供了一些工具，帮助您保护所有服务的数据，这些服务委托人已被授予访问您账户中资源的权限。

我们建议在资源策略中使用`aws:SourceArn`和`aws:SourceAccount`全局条件上下文密钥来限制为资源 AWS IoT Events 提供其他服务的权限。如果 `aws:SourceArn` 值不包含账户 ID，例如 Amazon S3 存储桶 ARN，您必须使用两个全局条件上下文密钥来限制权限。如果同时使用全局条件上下文密钥和包含账户 ID 的 `aws:SourceArn` 值，则 `aws:SourceAccount` 值和 `aws:SourceArn` 值中的账户在同一策略语句中使用时，必须使用相同的账户 ID。

如果您只希望将一个资源与跨服务访问相关联，请使用 aws:SourceArn。如果您想允许该账户中的任何资源与跨服务使用操作相关联，请使用。aws:SourceAccountaws:SourceArn值必须是与sts:AssumeRole请求关联的探测器模型或警报模型。

防范混淆代理问题最有效的方法是使用 aws:SourceArn 全局条件上下文键和资源的完整 ARN。如果不知道资源的完整 ARN，或者正在指定多个资源，请针对 ARN 未知部分使用带有通配符（*）的 aws:SourceArn 全局上下文条件键。例如 arn:aws:*iotevents*:*:*123456789012*:*。

以下示例说明了如何使用中的aws:SourceArn和aws:SourceAccount全局条件上下文键 AWS IoT Events 来防止出现混淆的副手问题。

主题

- 示例：安全访问 AWS IoT Events 探测器模型
- 示例：安全访问 AWS IoT Events 警报模型
- 示例：访问指定区域中的 AWS IoT Events 资源
- 示例：为配置日志选项 AWS IoT Events

示例：安全访问 AWS IoT Events 探测器模型

此示例演示如何创建一个 IAM 策略来安全地授予对中特定检测器模型的访问权限 AWS IoT Events。该策略使用条件来确保只有指定的 AWS 账户和 AWS IoT Events 服务才能担任该角色，从而增加了一层额外的安全性。在此示例中，角色只能访问名为的探测器模型*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"
                }
            }
        }
    ]
}
```

示例：安全访问 AWS IoT Events 警报模型

此示例演示如何创建允许 AWS IoT Events 安全访问警报模型的 IAM 策略。该策略使用条件来确保只有指定的 AWS 账户和 AWS IoT Events 服务才能担任该角色。

在此示例中，角色可以访问指定 AWS 账户内的任何警报模型，如警报模型 ARN 中的*通配符所示。`aws:SourceAccount`和`aws:SourceArn`条件共同作用，可以防止混乱的副手问题。

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/*"
                }
            }
        }
```

```
    ]
}
```

示例：访问指定区域中的 AWS IoT Events 资源

此示例演示如何配置 IAM 角色以访问特定 AWS 区域中的 AWS IoT Events 资源。通过在 IAM 策略 ARNs 中使用特定于区域的策略，您可以限制对不同地理区域的 AWS IoT Events 资源的访问。这种方法可以帮助维护多区域部署的安全性和合规性。此示例中的区域是*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:*"
                }
            }
        }
    ]
}
```

示例：为配置日志选项 AWS IoT Events

正确的日志记录对于监控、调试和审计 AWS IoT Events 应用程序非常重要。本节概述了中提供的日志记录选项 AWS IoT Events。

此示例演示如何配置允许将数据记录 AWS IoT Events 到日志的 IAM 角色。 CloudWatch 在资源 ARN 中使用通配符 (*) 可以全面记录整个 AWS IoT Events 基础架构。

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:*"
                }
            }
        }
    ]
}
```

对 AWS IoT Events 身份和访问进行故障排除

使用以下信息来帮助您诊断和修复在使用 AWS IoT Events 和 IAM 时可能遇到的常见问题。

主题

- 我无权在以下位置执行操作 AWS IoT Events
- 我无权执行 iam:PassRole
- 我想允许我以外的人 AWS 账户 访问我的 AWS IoT Events 资源

我无权在以下位置执行操作 AWS IoT Events

如果 AWS 管理控制台 告诉您您无权执行某项操作，则必须联系管理员寻求帮助。管理员是指提供用户名和密码的人员。

如果 mateojackson IAM 用户尝试使用控制台查看有关 *input* 的详细信息，但没有 iotevents:*ListInputs* 权限，则会出现以下示例错误。

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 iotevents:ListInputs on resource: my-example-input
```

在这种情况下，Mateo 请求他的管理员更新其策略，以允许他使用 iotevents:*ListInput* 操作访问 *my-example-input* 资源。

我无权执行 **iam:PassRole**

如果您收到一个错误，表明您无权执行 iam:PassRole 操作，则必须更新策略以允许您将角色传递给。 AWS IoT Events

有些 AWS 服务 允许您将现有角色传递给该服务，而不是创建新的服务角色或服务相关角色。为此，您必须具有将角色传递到服务的权限。

当名为 marymajor 的 IAM 用户尝试使用控制台在 AWS IoT Events中执行操作时，会发生以下示例错误。但是，服务必须具有服务角色所授予的权限才可执行此操作。Mary 不具有将角色传递到服务的权限。

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

在这种情况下，必须更新 Mary 的策略以允许她执行 iam:PassRole 操作。

如果您需要帮助，请联系您的 AWS 管理员。您的管理员是提供登录凭证的人。

我想允许我以外的人 AWS 账户 访问我的 AWS IoT Events 资源

您可以创建一个角色，以便其他账户中的用户或您组织外的人员可以使用该角色来访问您的资源。您可以指定谁值得信赖，可以代入角色。对于支持基于资源的策略或访问控制列表 (ACLs) 的服务，您可以使用这些策略向人们授予访问您的资源的权限。

请查阅以下主题以确定最佳选择：

- 要了解是否 AWS IoT Events 支持这些功能，请参阅如何 AWS IoT Events 与 IAM 配合使用。
- 要了解如何提供对您拥有的资源的访问权限 AWS 账户 ，请参阅 IAM 用户指南中的向您拥有 AWS 账户 的另一个 IAM 用户提供访问权限。
- 要了解如何向第三方提供对您的资源的访问权限 AWS 账户，请参阅 IAM 用户指南中的向第三方提供访问权限。 AWS 账户
- 要了解如何通过身份联合验证提供访问权限，请参阅《IAM 用户指南》中的为经过外部身份验证的用户（身份联合验证）提供访问权限。
- 要了解使用角色和基于资源的策略进行跨账户访问之间的差别，请参阅《IAM 用户指南》中的 IAM 中的跨账户资源访问。

监控 AWS IoT Events 以保持可靠性、可用性和性能

监控是维护 AWS 解决方案的可靠性、可用性和性能的重要组成部分。 AWS IoT Events 您应该从 AWS 解决方案的各个部分收集监控数据，以便在出现多点故障时可以更轻松地进行调试。在开始监控之前 AWS IoT Events，您应该创建一个包含以下问题的答案的监控计划：

- 监控目的是什么？
- 您将监控哪些资源？
- 监控这些资源的频率如何？
- 您将使用哪些监控工具？
- 谁负责执行监控任务？
- 出现错误时应通知谁？

下一步是通过测量不同时间和不同负载条件下的性能，为环境中的正常 AWS IoT Events 性能建立基准。在监控 AWS IoT Events时，存储历史监控数据，以便将此数据与当前性能数据进行比较，确定正常性能模式和性能异常，并设计解决问题的方法。

例如，如果您使用的是 Amazon EC2，则可以监控 CPU 利用率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、磁盘或减少网络流量。

主题

- 可供监控的可用工具 AWS IoT Events
- AWS IoT Events 使用 Amazon 进行监控 CloudWatch

- 使用记录 AWS IoT Events API 调用 AWS CloudTrail

可供监控的可用工具 AWS IoT Events

AWS 提供了各种可用于监控的工具 AWS IoT Events。您可以配置其中的一些工具来为您执行监控任务，但有些工具需要手动干预。建议您尽可能实现监控任务自动化。

自动监控工具

您可以使用以下自动监控工具来监视 AWS IoT Events 和报告何时出现问题：

- Amazon CloudWatch Logs — 监控、存储和访问来自 AWS CloudTrail 或其他来源的日志文件。有关更多信息，请参阅亚马逊 CloudWatch 用户指南中的使用亚马逊 CloudWatch控制面板。
- AWS CloudTrail 日志监控-在账户之间共享日志文件，通过将 CloudTrail 日志文件发送到“日志”来实时监控 CloudWatch 日志文件，用 Java 编写日志处理应用程序，并验证您的日志文件在传送后是否未更改。 CloudTrail有关更多信息，请参阅《AWS CloudTrail 用户指南》中的使用 CloudTrail 日志文件。

手动监控工具

监控 AWS IoT Events 的另一个重要部分是手动监控 CloudWatch 警报未涵盖的项目。 AWS IoT Events CloudWatch、和其他 AWS 控制台仪表板提供了 AWS 环境状态的 at-a-glance视图。我们建议您同时查看日志文件 AWS IoT Events。

- 控制 AWS IoT Events 台显示：
  - 探测器模型数
  - 探测器
  - 输入
  - 设置
- CloudWatch 主页显示：
  - 当前告警和状态
  - 告警和资源图表
  - 服务运行状况

  此外，您还可以使用 CloudWatch 执行以下操作：
  - 创建创建 CloudWatch仪表板以监控您关心的服务

- 绘制指标数据图，以排除问题并弄清楚趋势
- 搜索和浏览您的所有 AWS 资源指标
- 创建和编辑告警以接收问题通知

AWS IoT Events 使用 Amazon 进行监控 CloudWatch

开发或调试 AWS IoT Events 探测器模型时，你需要知道 AWS IoT Events 正在做什么，以及它遇到的任何错误。Amazon 会实时 CloudWatch 监控您的 AWS 资源和您运行 AWS 的应用程序。借 CloudWatch助，您可以全面了解资源使用情况、应用程序性能和运行状况。 在开发 AWS IoT Events 探测器模型时启用 Amazon CloudWatch 日志记录提供了有关如何为启用 CloudWatch 日志记录的信息 AWS IoT Events。要生成如下所示的日志，必须将详细级别设置为 “调试”，并提供一个或多个调试目标，即检测器型号名称和可选名称。KeyValue

以下示例显示了由生成的 CloudWatch DEBUG 级别的日志条目 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
    }
  ]
}
```

使用记录 AWS IoT Events API 调用 AWS CloudTrail

AWS IoT Events 与 AWS CloudTrail一项服务集成，该服务提供用户、角色或 AWS 服务在中执行的操作的记录 AWS IoT Events。 CloudTrail 将所有 API 调用捕获 AWS IoT Events 为事件，包括来自 AWS IoT Events 控制台的调用和对的代码调用 AWS IoT Events APIs。

如果您创建了跟踪，则可以允许将 CloudTrail 事件持续传输到 Amazon S3 存储桶，包括的事件 AWS IoT Events。如果您未配置跟踪，您仍然可以在 CloudTrail 控制台的 “事件历史记录” 中查看最新的事件。使用收集的信息 CloudTrail，您可以确定向哪个请求发出 AWS IoT Events、发出请求的 IP 地址、谁发出了请求、何时发出请求以及其他详细信息。

要了解更多信息 CloudTrail，请参阅《AWS CloudTrail 用户指南》。

AWS IoT Events 信息在 CloudTrail

CloudTrail 在您创建 AWS 账户时已在您的账户上启用。当活动发生在中时 AWS IoT Events，该活动与其他 AWS 服务 CloudTrail 事件一起记录在事件历史记录中。您可以在自己的 AWS 账户中查看、搜索和下载最近发生的事件。有关更多信息，请参阅使用 CloudTrail 事件历史记录。

要持续记录您 AWS 账户中的事件，包括的事件 AWS IoT Events，请创建跟踪。跟踪允许 CloudTrail 将日志文件传输到 Amazon S3 存储桶。默认情况下，当您在控制台中创建跟踪时，该跟踪将应用于所

有 AWS 区域。跟踪记录 AWS 分区中所有区域的事件，并将日志文件传送到您指定的 Amazon S3 存储桶。此外，您可以配置其他 AWS 服务，以进一步分析和处理 CloudTrail 日志中收集的事件数据。有关更多信息，请参阅:

- 为您的 AWS 账户创建跟踪
- CloudTrail 支持的服务和集成
- 配置 Amazon SNS 通知 CloudTrail
- 接收来自多个地区的 CloudTrail 日志文件和接收来自多个账户的 CloudTrail 日志文件

每个事件或日志条目都包含有关生成请求的人员信息。身份信息可帮助您确定以下内容：

- 请求是使用根用户凭证还是 IAM 用户凭证发出的。
- 请求是使用角色还是联合用户的临时安全凭证发出的。
- 请求是否由其他 AWS 服务发出。

有关更多信息，请参阅CloudTrail用户身份元素。 AWS IoT Events 操作记录在 AWS IoT Events API 参考中。

了解 AWS IoT Events 日志文件条目

跟踪是一种配置，允许将事件作为日志文件传输到您指定的 Amazon S3 存储桶。 AWS CloudTrail 日志文件包含一个或多个日志条目。事件代表来自任何来源的单个请求，包括有关请求的操作、操作的日期和时间、请求参数等的信息。 CloudTrail 日志文件不是公共 API 调用的有序堆栈跟踪，因此它们不会按任何特定的顺序出现。

在您的 AWS 账户中启用 CloudTrail 日志记录后，对 AWS IoT Events 操作发出的大多数 API 调用都将在 CloudTrail 日志文件中进行跟踪，这些调用与其他 AWS 服务记录一起写入日志文件。 CloudTrail根据时间段和文件大小决定何时创建和写入新文件。

每个日志条目都包含有关生成请求的人员的信息。日志条目中的用户身份信息可帮助您确定以下内容：

- 请求是使用根用户凭证还是 IAM 用户凭证发出的。
- 请求是使用角色还是联合用户的临时安全凭证发出的。
- 请求是否由其他 AWS 服务发出。

日志文件可以在 Amazon S3 存储桶中存储任意长时间，不过您也可以定义 Amazon S3 生命周期规则以自动存档或删除日志文件。默认情况下，系统将使用 Amazon S3 服务器端加密 (SSE) 对日志文件进行加密。

要在日志文件传送时收到通知，您可以配置 CloudTrail 为在传送新日志文件时发布 Amazon SNS 通知。有关更多信息，请参阅为 CloudTrail 配置 Amazon SNS 通知。

您还可以将来自多个 AWS 区域和多个 AWS 账户的 AWS IoT Events 日志文件聚合到单个 Amazon S3 存储桶中。

有关更多信息，请参阅接收来自多个区域的 CloudTrail 日志文件和接收来自多个账户的 CloudTrail 日志文件。

示例： DescribeDetector 操作用于 CloudTrail

以下示例显示了演示该`DescribeDetector`操作的 CloudTrail 日志条目。


```
{
  "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"
}
```

示例： CreateDetectorModel 操作用于 CloudTrail

以下示例显示了演示该`CreateDetectorModel`操作的 CloudTrail 日志条目。

```
{
  "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"
}
```

示例： CreateInput 操作用于 CloudTrail

以下示例显示了演示该`CreateInput`操作的 CloudTrail 日志条目。

```
{
  "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"
}
```

示例： DeleteDetectorModel 操作用于 CloudTrail

以下示例显示了演示该`DeleteDetectorModel`操作的 CloudTrail 日志条目。

```
{
  "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"
}
```

示例： DeleteInput 操作用于 CloudTrail

以下示例显示了演示该`DeleteInput`操作的 CloudTrail 日志条目。

```
{
  "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"
}
```

示例： DescribeDetectorModel 操作用于 CloudTrail

以下示例显示了演示该`DescribeDetectorModel`操作的 CloudTrail 日志条目。

```
{
  "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"
}
```

示例： DescribeInput 操作用于 CloudTrail

以下示例显示了演示该`DescribeInput`操作的 CloudTrail 日志条目。

```
{
  "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"
}
```

示例： DescribeLoggingOptions 操作用于 CloudTrail

以下示例显示了演示该DescribeLoggingOptions操作的 CloudTrail 日志条目。

```
{
  "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"
}
```

示例： ListDetectorModels 操作用于 CloudTrail

以下示例显示了演示该`ListDetectorModels`操作的 CloudTrail 日志条目。

```
{
  "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"
}
```

示例： ListDetectorModelVersions 操作用于 CloudTrail

以下示例显示了演示该`ListDetectorModelVersions`操作的 CloudTrail 日志条目。

```
{
  "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"
}
```

示例： ListDetectors 操作用于 CloudTrail

以下示例显示了演示该`ListDetectors`操作的 CloudTrail 日志条目。

```
{
  "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"
}
```

示例： ListInputs 操作用于 CloudTrail

以下示例显示了演示该`ListInputs`操作的 CloudTrail 日志条目。

```
{
  "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"
}
```

示例： PutLoggingOptions 操作用于 CloudTrail

以下示例显示了演示该PutLoggingOptions操作的 CloudTrail 日志条目。

```
{
  "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"
}
```

示例： UpdateDetectorModel 操作用于 CloudTrail

以下示例显示了演示该UpdateDetectorModel操作的 CloudTrail 日志条目。

```
{
  "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"
}
```

示例： UpdateInput 操作用于 CloudTrail

以下示例显示了演示该UpdateInput操作的 CloudTrail 日志条目。

```
{
  "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"
}
```

示例： BatchPutMessage 操作用于 CloudTrail

AWS IoT Events 可以使用 CloudTrail 集成进行数据平面 API 日志记录。此示例通过BatchPutMessage操作添加了有关数据事件的详细信息。

```
{
            "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"
            }
        },
```

合规性验证 AWS IoT Events

要了解是否属于特定合规计划的范围，请参阅AWS 服务 “按合规计划划分的范围” ”，然后选择您感兴趣的合规计划。 AWS 服务 有关一般信息，请参阅AWS 合规计划AWS。

您可以使用下载第三方审计报告 AWS Artifact。有关更多信息，请参阅中的 “下载报告” 中的 “ AWS Artifact。

您在使用 AWS 服务 时的合规责任取决于您的数据的敏感性、贵公司的合规目标以及适用的法律和法规。有关您在使用时的合规责任的更多信息 AWS 服务，请参阅AWS 安全文档。

韧性在 AWS IoT Events

AWS 全球基础设施是围绕 AWS 区域和可用区构建的。 AWS 区域提供多个在物理上独立且隔离的可用区，这些可用区通过延迟低、吞吐量高且冗余性高的网络连接在一起。利用可用区，您可以设计和操作在可用区之间无中断地自动实现故障转移的应用程序和数据库。与传统的单个或多个数据中心基础架构相比，可用区具有更高的可用性、容错性和可扩展性。

有关 AWS 区域和可用区的更多信息，请参阅AWS 全球基础设施。

中的基础设施安全 AWS IoT Events

作为一项托管服务 AWS IoT Events ，受 AWS 全球网络安全的保护。有关 AWS 安全服务以及如何 AWS 保护基础设施的信息，请参阅AWS 云安全。要使用基础设施安全的最佳实践来设计您的 AWS 环境，请参阅 S AWS ecurity Pillar Well-Architected Fram ework 中的基础设施保护。

您可以使用 AWS 已发布的 API 调用 AWS IoT Events 通过网络进行访问。客户端必须支持以下内容：

- 传输层安全性协议（ TLS ）。我们要求使用 TLS 1.2，建议使用 TLS 1.3。

- 具有完全向前保密（PFS）的密码套件，例如 DHE（临时 Diffie-Hellman）或 ECDHE（临时椭圆曲线 Diffie-Hellman）。大多数现代系统（如 Java 7 及更高版本）都支持这些模式。

AWS AWS IoT Events 资源的服务配额

该AWS 一般参考 指南提供了 AWS 账户 AWS IoT Events 的默认配额。除非另有说明，否则每个配额都是按 AWS 地区划分的。有关更多信息，请参阅AWS 一般参考 指南 中的AWS IoT Events 终点和配额 及 AWS 服务配额。

要请求提高服务配额，请在支持中心控制台中提交支持案例。有关更多信息，请参阅《Service Quotas 用户指南》中的请求增加配额。

> **Note**
>
> - 在账户中，探测器模型和输入的所有名称必须是唯一的。
> - 在创建探测器模型和输入之后，您无法更改它们的名称。

标记您的资源 AWS IoT Events

为了帮助您管理和组织探测器模型和输入，您可以选择将自己的元数据以标签的形式分配给其中每个资源。本部分介绍标签并说明如何创建标签。

标签基本知识

标签使您能够以不同的方式对 AWS IoT Events 资源进行分类，例如按用途、所有者或环境进行分类。这在您有许多相同类型的资源时会非常有用。您可以根据分配到特定资源的标签来快速识别该资源。

每个标签都包含您定义的一个键和一个可选值。例如，您可为输入定义一系列标签，以帮助您按类型追踪发送这些输入的设备。我们建议您为每类资源创建一组可满足您的需求的标签键。使用一组连续的标签键，管理 资源时会更加轻松。

您可以根据您添加或应用的标签搜索和筛选资源，使用标签分类和追踪成本，还可以使用标签控制对资源的访问权限，如AWS IoT 《开发者指南》中的通过 IAM 策略使用标签所述。

为便于使用，中的标签编辑器 AWS 管理控制台 提供了一种集中、统一的方式来创建和管理标签。有关更多信息，请参阅《标签 AWS 资源和标签编辑器用户指南》中的标签编辑器入门。

您也可以使用 AWS CLI 和 AWS IoT Events API 处理标签。创建标签时，您可使用以下命令中的"Tags"字段，将标签与探测器模型和输入值关联：

- CreateDetectorModel
- CreateInput

您可以使用以下命令为支持标记的现有资源添加、修改或删除标签：

- TagResource
- ListTagsForResource
- UntagResource

您可以修改标签的键和值，还可以随时删除资源的标签。您可以将标签的值设为空的字符串，但是不能将其设为空值。如果添加的标签的键与该资源上现有标签的键相同，新值就会覆盖旧值。如果删除资源，则所有与资源相关的标签都将被删除。

有关更多信息，请参阅为资源添加标签 AWS 的最佳实践

标签限制

下面是适用于 标签的基本限制：

- 每个资源的标签数上限：50
- 最大键长度 – 127 个 Unicode 字符 (采用 UTF-8 格式)
- 最大值长度 – 255 个 Unicode 字符 (采用 UTF-8 格式)
- 标签键和值区分大小写。
- 请勿在标签名称或值中使用"aws:"前缀，因为它已保留供 AWS 使用。您无法编辑或删除带此前缀的标签名称或值。具有此前缀的标签不计入每个资源的标签数限制。
- 如果标签方案针对多个服务和资源使用，请记得其他服务可能对支持的字符有限制。通常允许使用的字符包括：可用 UTF-8 格式表示的字母、空格和数字以及以下特殊字符：+ - = . _ : / @。

在 IAM 策略中使用标签

可以在用于 AWS IoT Events API 操作的 IAM 策略中应用基于标签的资源级权限。这可让您更好地控制用户可创建、修改或使用哪些资源。

在 IAM 策略中将 Condition 元素（也称作 Condition 块）与以下条件上下文键和值结合使用来基于资源标签控制用户访问（权限）：

- 使用 `aws:ResourceTag/<tag-key>: <tag-value>` 可允许或拒绝带特定标签的资源上的用户操作。
- 使用 `aws:RequestTag/<tag-key>: <tag-value>` 可要求在发出创建或修改允许标签的资源的 API 请求时使用（或不使用）特定标签。
- 使用 `aws:TagKeys: [<tag-key>, ...]` 可要求在发出创建或修改允许标签的资源的 API 请求时使用（或不使用）一组特定标签键。

> **Note**
>
> IAM 策略中的条件上下文键和值仅适用于能够标记的资源的标识符是必需参数的那些 AWS IoT Events 操作。

AWS Identity and Access Management 用户指南中的使用标签控制访问权限包含有关使用标签的额外信息。该指南的 IAM JSON 策略参考部分包含 IAM 中的 JSON 策略的元素、变量和评估逻辑的详细语法、描述和示例。

以下策略示例应用两个基于标签的限制。受此策略限制的用户：

- 无法为资源提供标签“env=prod”（在示例中，请参阅行 `"aws:RequestTag/env" : "prod"`
- 无法修改或访问具有现有标签“env=prod”的资源（在示例中，请参阅行 `"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": "*"
        }
    ]
}
```

您还可以为给定标签键指定多个标签值，方法是将它们括在列表中，如下所述。

```
        "StringEquals" : {
          "aws:ResourceTag/env" : ["dev", "test"]
        }
```

> **Note**
>
> 如果您基于标签允许或拒绝用户访问资源，则必须考虑显式拒绝用户对相同资源添加或删除这些标签的能力。否则，用户可能通过修改资源标签来绕过您的限制并获得资源访问权限。

故障排除 AWS IoT Events

本疑难解答指南提供了您在使用时可能遇到的常见问题的解决方案 AWS IoT Events。浏览主题以识别和解决检测事件、访问数据、权限、服务集成、设备配置等方面的问题。本指南提供有关 AWS IoT Events 控制台、API、CLI、错误、延迟和集成的故障排除建议，旨在快速解决您的问题，以便您可以构建可靠且可扩展的事件驱动型应用程序。

主题

常见 AWS IoT Events 问题和解决方案

请参阅以下部分，对错误进行故障排除，并找到解决问题的可能解决方案 AWS IoT Events。

错误

探测器模型创建错误

我在尝试创建探测器模型时出错。

解决方案

创建探测器模型时，您必须考虑以下限制。

- 每个 action 字段中只允许执行一个操作。
- 对于transitionEvents，condition 是必填项。对于OnEnter、OnInput 和 OnExit 事件，它是选填项。
- 如果condition字段为空，则条件表达式的计算结果等同于 true。
- 条件表达式的计算结果应为布尔值。如果结果不是布尔值，则它等同于 false 且不会触发 actions 或转换为事件中指定的nextState。

有关更多信息，请参阅 AWS IoT Events 探测器型号限制和限制。

从已删除的探测器模型获取更新

我几分钟前更新或删除了一个探测器模型，但我仍能通过 MQTT 消息或 SNS 提醒从旧探测器模型获取状态更新。

解决方案

如果您更新、删除或重新创建探测器模型（参见 UpdateDetectorModel），则在删除所有探测器实例并使用新模型之前，会有一段延迟。在此期间，输入可能会继续由先前版本的探测器模型实例处理。您可能会继续收到由先前探测器型号定义的提醒。请至少等待 7 分钟，然后再重新检查更新或者报告错误。

操作触发失败（满足条件时）

当满足条件时，探测器无法触发操作或过渡至新状态。

解决方案

验证探测器的条件表达式的计算结果是否为布尔值。如果结果不是布尔值，则它等同于 false 且不会触发 action 或转换为事件中指定的nextState。有关更多信息，请参阅条件表达式语法。

操作触发失败（违反阈值时）

当条件表达式中的变量达到指定值时，探测器不会触发操作或事件转换。

解决方案

如果您为onInput、onEnter、或onExit更新setVariable，则在当前处理周期内评估任何condition时都不会使用新值。相反，原始值会在当前周期完成前一直使用。您可通过按探测器模型定义设置evaluationMethod参数，以更改此行为。如果将evaluationMethod设置为SERIAL，则按事件的定义顺序更新变量并评估事件条件。如果将evaluationMethod设置为BATCH（默认），则仅在评估所有事件条件后才会更新变量并执行事件。

状态用途不正确

当我尝试通过BatchPutMessage向输入发送消息时，探测器进入错误的状态。

解决方案

如果您使用BatchPutMessage向输入发送多条消息，则无法保证消息或输入的处理顺序。为保证顺序，请一次发送一条消息，然后每次等待BatchPutMessage以确认成功。

连接消息

当我尝试调用 API 时收到了('Connection aborted.', error(54, 'Connection reset by peer'))错误。

解决方案

验证 OpenSSL 是否使用 TLS 1.1 或更高版本建立连接。这应当是大多数 Linux 发行版或 Windows 7 及更高版本下的默认设置。macOS 用户可能需要升级 OpenSSL。

InvalidRequestException 消息

当我尝试打电话 InvalidRequestException 时我明白了CreateDetectorModel，UpdateDetectorModel APIs.

解决方案

查看以下内容，以帮助解决问题。有关更多信息，请参阅CreateDetectorModel和UpdateDetectorModel。

- 确保不要同时使用seconds和durationExpression作为SetTimerAction的参数。
- 请确保您的durationExpression字符串表达式有效。字符串表达式可以包含数字、变量 ($variable.<variable-name>) 或输入值 ($input.<input-name>.<path-to-datum>)。

Amazon CloudWatch 日志`action.setTimer`错误

您可以设置 Amazon CloudWatch Logs 来监控 AWS IoT Events 探测器模型实例。以下是您在使用时生成的 AWS IoT Events常见错误`action.setTimer`。

- 错误：无法将名为*<timer-name>*的计时器的持续时间表达式求值为数字。

解决方案

确保您的`durationExpression`字符串表达式可以转换为数字。不允许使用其他数据类型，如布尔值。

- 错误：名为*<timer-name>*的计时器的持续时间表达式的计算结果大于 31622440。为确保准确性，请确保您的持续时间表达式指介于 60 和 31622400 之间的值。

解决方案

确保您的计时器持续时间小于或等于 31622400 秒。持续时间的计算结果向下舍入为最接近的整数。

- 错误：名为*<timer-name>*的计时器的持续时间表达式的计算结果小于 60。为确保准确性，请确保您的持续时间表达式指介于 60 和 31622400 之间的值。

解决方案

确保计时器的持续时间大于或等于 60 秒。持续时间的计算结果向下舍入为最接近的整数。

- 错误：名为*<timer-name>*的计时器的持续时间表达式的结果无法计算。检查变量名称、输入名称和数据路径，以确保引用了现有变量和输入值。

解决方案

确保您的字符串表达式引用了现有的变量和输入值。字符串表达式可以包含数字、变量 (`$variable.`*variable-name*) 和输入值 (`$input.`*input-name*`.`*path-to-datum*)。

- 错误：无法设置名为<timer-name>的计时器。请检查您的持续时间表达式，然后重试。

解决方案

请查看SetTimerAction操作以确保您指定了正确的参数，然后再次设置计时器。

有关更多信息，请参阅开发 AWS IoT Events 探测器模型时启用 Amazon CloudWatch 日志记录。

Amazon CloudWatch 有效负载错误

您可以设置 Amazon CloudWatch Logs 来监控 AWS IoT Events 探测器模型实例。以下是您在配置操作负载时生成的 AWS IoT Events常见错误和警告。

- 错误：我们无法求解您的操作表达式。确保变量名称、输入名称和数据路径引用了现有变量和输入值。此外，请验证有效负载的大小是否小于 1 KB，即有效负载的最大允许大小。

解决方案

确保输入正确的变量名称、输入名称和数据路径。如果操作有效负载大于 1 KB，您也可能会收到此错误消息。

- 错误：我们无法解析*<action-type>*有效负载的内容表达式。输入语法正确的内容表达式。

解决方案

内容表达式可以包含字符串 ('*string*')、变量 ($variable.*variable-name*)、输入值 ($input.*input-name*.*path-to-datum*)、字符串串联，以及包含 ${}的字符串。

- 错误：您的负载表达式 {*expression*} 无效。定义的负载类型为 JSON，因此您必须指定 AWS IoT Events 一个计算结果为字符串的表达式。

解决方案

如果指定的负载类型为 JSON，则 AWS IoT Events 首先检查服务是否可以将您的表达式计算为字符串。计算结果不得为布尔值或数字。如果验证失败，您可能会收到此错误消息。

- 警告：此操作已执行，但我们无法将操作负载的内容表达式求值为有效的 JSON。定义的有效负载类型为 JSON。

解决方案

如果您将有效负载类型定义为，请确保 AWS IoT Events 可以将操作负载的内容表达式评估为有效的 JSON `JSON`。 AWS IoT Events 即使 AWS IoT Events 无法将内容表达式评估为有效的 JSON，也会运行操作。

有关更多信息，请参阅开发 AWS IoT Events 探测器模型时启用 Amazon CloudWatch 日志记录。

数据类型不兼容

消息：在以下表达式中找到了不兼容的<inferred-types>数据类型
[<reference>]：<expression>

解决方案

收到此错误的原因可能是以下任一原因：

- 您引用的计算结果与表达式中的其他操作数不兼容。
- 不支持传递至函数的自变量的类型。

当您在表达式中使用引用时，请检查以下内容：

- 当您将引用值作为一个或多个运算符的操作数时，确保您引用的所有数据类型均兼容。

  例如，在以下表达式中，整数 2 是 == 和 && 运算符的操作数。为确保操作数兼容，$variable.testVariable + 1 和 $variable.testVariable必须引用整数或小数。

  此外，整数 1 是运算符+的操作数。因此，$variable.testVariable 必须引用整数或小数。

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

- 当您使用引用作为传递给函数的自变量时，请确保该函数支持您引用的数据类型。

  例如，以下 timeout("*time-name*") 函数需要带有双引号的字符串作为自变量。如果使用引用作为*timer-name*值，则必须使用双引号引用字符串。

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

  > **Note**
  >
  > 对于convert(*type*, *expression*)函数，如果您使用引用作为*type*值，则引用的计算结果必须是StringDecimal、或Boolean。

有关更多信息，请参阅 AWS IoT Events 表达式中输入和变量的参考。

向发送消息失败 AWS IoT Events

消息：无法向物联网事件发送消息

解决方案

出现此错误的原因如下：

- 输入消息有效负载不包含 `Input attribute Key`。
- `Input attribute Key`与输入定义中指定的 JSON 路径不同。
- 输入消息与 AWS IoT Events 输入中定义的架构不匹配。

> **Note**
>
> 其他服务的数据摄取也会出现失败。

Example

例如 AWS IoT Core，在中， AWS IoT 规则将失败，并显示以下消息 `Verify the Input Attribute key.`

要解决这个问题，请确保输入有效载荷消息架构符合 AWS IoT Events 输入定义并且`Input attribute Key`位置匹配。有关更多信息，请参阅在中为模型创建输入 AWS IoT Events以了解如何定义 AWS IoT Events 输入。

通过在中运行分析对探测器模型进行故障排除 AWS IoT Events

AWS IoT Events 无需向探测器模型发送输入数据即可分析您的探测器模型并生成分析结果。 AWS IoT Events 执行本节所述的一系列分析以检查您的探测器模型。此高级故障排除解决方案还汇总了诊断信息，包括严重级别和位置，以便您可以快速发现和修复探测器模型中的潜在问题。有关您的探测器模型的诊断错误类型和消息的更多信息，请参阅 的探测器模型分析和诊断信息 AWS IoT Events。

您可以使用 AWS IoT Events 控制台、API、AWS Command Line Interface (AWS CLI) 或 AWS SDK 查看检测器模型分析产生的诊断错误消息。

> **Note**
>
> - 在发布探测器模型之前，必须修复所有错误。

- 我们建议您在生产环境中使用探测器模型之前，先查看警告并采取必要的行动。否则，探测器模型可能无法按预期工作。
- 最多可拥有 10 个分析同时处于 RUNNING 状态。

要了解如何分析探测器模型，请参阅 分析探测器模型 AWS IoT Events （控制台） 或 在 AWS IoT Events (AWS CLI) 中分析探测器模型。

主题

的探测器模型分析和诊断信息 AWS IoT Events

探测器模型分析收集以下诊断信息：

- 级别 - 分析结果的严重性级别。根据严重性级别，分析结果分为三大类：
  - 信息 (INFO) - 信息结果告诉您探测器模型中的一个重要字段。这类结果通常不需要立即采取行动。
  - 警告 (WARNING) - 警告结果会提请特别注意可能会导致探测器模型出现问题的字段。我们建议您在生产环境中使用探测器模型之前，先查看警告并采取必要的行动。否则，探测器模型可能无法按预期工作。
  - 错误 (ERROR - 错误结果会通知您探测器模型中发现的问题。当您尝试发布探测器模型时， AWS IoT Events 会自动执行这组分析。在发布探测器模型之前，必须修复所有错误。
- 位置 - 包含可用于在探测器模型中定位可供分析结果引用的字段的信息。位置通常包括状态名称、过渡事件名称、事件名称和表达式（例如 `in state TemperatureCheck in onEnter in event Init in action setVariable`）。
- 类型 - 分析结果的类型。分析类型分为以下几类：
  - `supported-actions`— AWS IoT Events 可以在检测到指定事件或过渡事件时调用操作。您可以定义内置操作以使用计时器或设置变量，或者将数据发送到其他 AWS 服务。在 AWS 提供服务的 AWS 区域，您必须指定适用于其他 AWS 服务的操作。
  - `service-limits`— 服务配额，也称为限制，是您的 AWS 账户中服务资源或操作的最大或最小数量。除非另有说明，否则每个配额都是特定于区域的。根据您的业务需求，您可以更新探测器模

型以避免遇到限制或申请增加配额。您可以请求增加某些配额，但其他一些配额无法增加。有关更多信息，请参阅 配额。

- **structure** — 探测器模型必须具有所有必需的组件，例如状态，并遵循 AWS IoT Events 支持的结构。探测器模型必须至少具有一种状态和一个能评估输入数据以检测重要事件的条件。当检测到事件时，探测器模型会过渡到下一个状态并可以调用操作。这些事件称为过渡事件。过渡事件必须引导下一个要进入的状态。
- **expression-syntax**— AWS IoT Events 提供了多种在创建和更新探测器模型时指定值的方法。您可以在表达式中使用文字、运算符、函数、引用和替代模板。您可以使用表达式来指定文字值，也 AWS IoT Events 可以在指定特定值之前对表达式求值。您的表达式必须遵循所需的语法。有关更多信息，请参阅 用于筛选、转换和处理事件数据的表达式。

  中的探测器模型表达式 AWS IoT Events 可以引用特定的数据或资源。

  - **data-type**— AWS IoT Events 支持整数、小数值、字符串和布尔数据类型。如果 AWS IoT Events 可以在表达式求值期间自动将一种数据类型的数据转换为另一种数据类型，则这些数据类型是兼容的。

    > **Note**
    > - 整数和小数值是唯一受 AWS IoT Events支持的兼容数据类型。
    > - AWS IoT Events 无法计算算术表达式，因为 AWS IoT Events 无法将整数转换为字符串。

  - **referenced-data** — 必须先定义探测器模型中引用的数据，然后才能使用这些数据。例如，如果要向 DynamoDB 表发送数据，则必须先定义一个引用表名的变量，然后才能在表达式 (`$variable.TableName`) 中使用该变量。
  - **referenced-resource** — 探测器模型使用的资源必须可用。您必须先定义资源，然后才能使用它们。例如，您要创建探测器模型以监控温室的温度。必须先定义一个输入 (`$input.TemperatureInput`)，将传入的温度数据路由到探测器模型，然后才能使用 `$input.TemperatureInput.sensorData.temperature` 来引用温度。

请参阅以下部分，对错误进行故障排除，并通过对探测器模型的分析找到可能的解决方案。

对中的探测器模型错误进行故障排除 AWS IoT Events

上述类型错误提供有关探测器模型的诊断信息，并与您可能检索到的消息相对应。使用这些消息和建议的解决方案来排除探测器模型的错误。

消息和解决方案

- Location
- supported-actions
- service-limits
- structure
- expression-syntax
- data-type
- referenced-data
- referenced-resource

`Location`

包含 `Location` 相关信息的分析结果对应以下错误消息：

- 消息 - 包含有关分析结果的其他信息。这可以是信息、警告或错误消息。

  解决方案：如果您指定的操作 AWS IoT Events 当前不支持，则可能会收到此错误消息。有关受支持的操作列表，请参阅支持在中接收数据和触发操作的操作 AWS IoT Events。

`supported-actions`

包含 `supported-actions` 相关信息的分析结果对应以下错误消息：

- 消息：动作定义中存在无效的操作类型：*`action-definition`*。

  解决方案：如果您指定的操作 AWS IoT Events 当前不支持，则可能会收到此错误消息。有关受支持的操作列表，请参阅支持在中接收数据和触发操作的操作 AWS IoT Events。
- 消息： DetectorModel 定义有*`aws-service`*操作，但该地区不支持该*`aws-service`*服务*`region-name`*。

  解决方案：如果您指定的操作受支持，但该操作在您当前的地区不可用 AWS IoT Events，则您可能会收到此错误消息。当您尝试向该地区不可用的 AWS 服务发送数据时，可能会发生这种情况。您还必须为两者 AWS IoT Events 以及您正在使用的 AWS 服务选择相同的区域。

`service-limits`

包含 `service-limits` 相关信息的分析结果对应以下错误消息：

- 消息：有效载荷中允许的内容表达式超过了*event-name*处于状态的事件*content-expression-size*字节数限制*state-name*。

  解决方案：如果操作负载的内容表达式大于 1024 字节，则可能会收到此错误消息。负载的内容表达式大小最多可为 1024 字节。

- 消息：探测器模型定义中允许的状态数超过了限制*states-per-detector-model*。

  解决方案：如果您的探测器模型的状态超过 20 个，则可能会收到此错误消息。探测器模型最多可拥有 20 个状态。

- 消息：计时器的持续时间*timer-name*应至少为*minimum-timer-duration*几秒钟。

  解决方案：如果计时器的持续时间少于 60 秒，则可能会收到此错误消息。我们建议计时器的持续时间应在 60 到 31622400 秒之间。如果您指定计时器持续时间的表达式，则持续时间表达式的计算结果向下舍入为最接近的整数。

- 消息：每个事件允许的操作数量超过了探测器模型定义*actions-per-event*中的限制

  解决方案：如果事件的操作超过 10 个，您可能会收到此错误消息。对于探测器模型中的每个事件，最多可有 10 个操作。

- 消息：每个状态允许的过渡事件数量超过了探测器模型定义*transition-events-per-state*中的限制。

  解决方案：如果一个状态有超过 20 个过渡事件，则可能会收到此错误消息。探测器模型中的每个状态最多可拥有 20 个过渡事件。

- 消息：每个状态允许的事件数超过了探测器模型定义*events-per-state*中的限制

  解决方案：如果一个状态有超过 20 个事件，您可能会收到此错误消息。探测器模型中的每个状态最多可拥有 20 个事件。

- 消息：可以与单个输入关联的最大探测器模型数量可能已超过限制。输入*input-name*用于*detector-models-per-input*探测器模型路径。

  解决方案：如果您尝试将输入路由到超过 10 个探测器模型，则可能会收到此警告消息。您最多可以将 10 个不同的探测器模型与单个探测器模型相关联。

`structure`

包含 `structure` 相关信息的分析结果对应以下错误消息：

- 消息：动作可能只定义了一种类型，但发现了一个带有*number-of-types*类型的动作。请分成单独的操作。

  解决方案：如果您使用 API 操作来创建或更新探测器模型，在单个字段中指定了两个或多个操作，则可能会收到此错误消息。您可以定义一组 `Action` 对象。确保将每个操作定义为一个单独的对象。

- 消息： TransitionEvent*transition-event-name*过渡到不存在的状态。*state-name*

  解决方案：如果找 AWS IoT Events 不到过渡事件引用的下一个状态，则可能会收到此错误消息。确保定义了下一个状态并输入了正确的状态名称。

- 消息：有一个共享的状态名称： DetectorModelDefinition 已找到*state-name*带有*number-of-states*重复内容的状态。

  解决方案：如果您对一个或多个状态使用相同的名称，则可能会收到此错误消息。确保为探测器模型中的每个状态指定唯一的名称。状态名称必须具有 1-128 个字符。有效字符：a-z、A-Z、0-9、_（下划线）和 -（连字符）。

- 消息：定义与定义的状态 initialStateName *initial-state-name*不对应。

  解决方案：如果初始状态名称不正确，您可能会收到此错误消息。在输入到达之前，探测器模型将保持初始（开始）状态。输入到达后，探测器模型会立即过渡到下一个状态。确保初始状态名称是已定义状态的名称，并且您输入了正确的名称。

- 消息：探测器模型定义必须在一个条件中使用至少一个输入。

  解决方案：如果您未在条件中指定输入，则可能会收到此错误。您必须在至少一个条件下使用至少一个输入。否则， AWS IoT Events 不评估传入的数据。

- 消息：只能在中设置秒数和持续时间表达式。 SetTimer

  解决方案：如果您为计时器同时使用 `seconds` 和 `durationExpression`，则可能会收到此错误消息。请确保使用 `seconds` 或 `durationExpression` 作为 `SetTimerAction` 的参数。有关更多信息，请参阅《AWS IoT Events API Reference》中的 SetTimerAction。

- 消息：探测器模型中的操作无法访问。检查启动操作的条件。

  解决方案：如果探测器模型中的某个操作无法访问，则该事件的条件评估为 false。检查包含操作的事件的条件，确保其计算结果为 true。当事件的条件计算为 true 时，操作应变得可访问。

- 消息：正在读取输入属性，但这可能是由计时器过期引起的。

  解决方案：当出现以下任一情况时，可以读取输入属性的值：

- 已收到新的输入值。
- 当探测器中的计时器已过期。

为确保仅在收到输入的新值时才对输入属性进行评估，请在条件中包括对 `triggerType("Message")` 函数的调用，如下所示：

探测器模型中正在评估的原始条件：

```
if ($input.HeartBeat.status == "OFFLINE")
```

类似于以下内容：

```
if ( triggerType("MESSAGE") &&  $input.HeartBeat.status == "OFFLINE")
```

其中，对 `triggerType("Message")` 函数的调用是在条件中提供的初始输入之前进行的。通过使用这种技术，`triggerType("Message")` 函数的计算结果将为 true 并满足接收新输入值的条件。有关 `triggerType` 函数用法的更多信息，请在AWS IoT Events 《开发人员指南》的 "表达式" 部分中搜索 `triggerType`

- 消息：您的探测器模型中的状态无法访问。检查会导致过渡到所需状态的条件。

  解决方案：如果探测器模型中的某个状态不可达，则导致传入过渡到该状态的条件评估为 false。检查探测器模型中传入的过渡到该不可达状态的条件是否评估为 true，这样所需的状态就可变为可访问。

- 消息：计时器即将到期可能会导致发送意外数量的消息。

  解决方案：为防止您的探测器模型因计时器已过期而进入发送意外数量的消息的无限状态，请考虑在探测器模型的条件下使用对 `triggerType("Message")` 函数的调用，如下所示：

  探测器模型中正在评估的原始条件：

  ```
  if (timeout("awake"))
  ```

  会转化为类似于以下内容的条件：

  ```
  if (triggerType("MESSAGE") && timeout("awake"))
  ```

  其中，对 `triggerType("Message")` 函数的调用是在条件中提供的初始输入之前进行的。

此更改可防止在探测器中启动计时器操作，从而防止发送无限循环的消息。有关如何在探测器中使用计时器操作的更多信息，请参阅《AWS IoT Events 开发者指南》的 “使用内置操作” 页面

expression-syntax

包含 expression-syntax 相关信息的分析结果对应以下错误消息：

- 消息：您的负载表达式 {*expression*} 无效。定义的负载类型为 JSON，因此您必须指定 AWS IoT Events 一个计算结果为字符串的表达式。

  解决方案：如果指定的负载类型为 JSON，则 AWS IoT Events 首先检查服务是否可以将您的表达式计算为字符串。计算结果不得为布尔值或数字。如果验证不成功，您可能会收到此错误。

- 消息：SetVariableAction.value 必须是表达式。无法解析值 “” *variable-value*

  解决方案：您可以使用 SetVariableAction 来定义具有 name 和 value 的变量。value 可以是字符串、数字或布尔值。您也可以为 value 指定表达式。有关更多信息 SetVariableAction，请参阅 AWS IoT Events API 参考中的。

- 消息：我们无法解析你对 DynamoDB 操作的属性 (*attribute-name*) 表达式。使用正确的语法输入表达式。

  解决方案：必须对 DynamoDBAction. 替换模板中的所有参数使用表达式。有关更多信息，请参阅 AWS IoT Events API 参考DBAction中的 Dynamo。

- 消息：我们无法解析你对 Dynamo 操作的 TableName 表达式。DBv2 使用正确的语法输入表达式。

  解决方案：DynamoDBv2Action 中的 tableName 必须是字符串。必须使用 tableName 的表达式。这些表达式接受文字、运算符、函数、引用和替代模板。有关更多信息，请参阅 AWS IoT Events API 参考中的 Dynamo DBv2 操作。

- 消息：我们无法将您的表达式评估为有效的 JSON。Dynamo DBv2 操作仅支持 JSON 有效负载类型。

  解决方案：DynamoDBv2 的负载类型必须为 JSON。确保它 AWS IoT Events 可以将您的内容表达式评估为有效的 JSON。有关更多信息，请参阅 AWS IoT Events API 参考中的 Dynamo DBv2 操作。

- 消息：我们无法解析你的内容表达式的有效载荷。*action-type*输入语法正确的内容表达式。

  解决方案：内容表达式可以包含字符串 ('*string*')、变量 ($variable。 *variable-name*)、输入值（ $input。 *input-name*。 *path-to-datum*)、字符串连接和包含. 的字符串。${}

- 消息：自定义负载必须为非空。

  解决方案：如果您为操作选择了自定义负载，但没有在 AWS IoT Events 控制台中输入内容表达式，则可能会收到此错误消息。如果选择自定义负载，则必须在自定义负载下输入内容表达式。有关更多信息，请参阅《AWS IoT Events API 参考》中的有效负载。

- 消息：无法解析计时器 “*duration-expression*” 的持续时间表达式 “*timer-name*”。

  解决方案：计时器持续时间表达式的计算结果必须是介于 60—31622400 之间的值。持续时间的计算结果向下舍入为最接近的整数。

- 消息：无法解析表达式 “*expression*” *action-name*

  解决方案：如果指定操作的表达式语法不正确，则可能会收到此消息。请确保使用正确的语法输入表达式。有关更多信息，请参阅 用于筛选设备数据和定义操作的语法 AWS IoT Events。

- 消息：IotSitewiseAction无法解析你*fieldName*的 for。必须在表达式中使用正确的语法。

  解决方案：如果 AWS IoT Events 无法解析您的 for，则可能会收到此错误*fieldName*。IotSitewiseAction确保*fieldName*使用 AWS IoT Events 可以解析的表达式。有关更多信息，请参阅《AWS IoT Events API Reference》中的 IotSiteWiseAction。

data-type

包含 data-type 相关信息的分析结果对应以下错误消息：

- 消息：计时*duration-expression*器的持续时间表达式*timer-name*无效，它必须返回一个数字。

  解决方案：如果 AWS IoT Events 无法将计时器的持续时间表达式计算为数字，则可能会收到此错误消息。确保您的 durationExpression 可转换为数字。不支持其他数据类型，例如布尔值。

- 消息：表达式*condition-expression*不是有效的条件表达式。

  解决方案：如果 AWS IoT Events 无法将您的condition-expression值计算为布尔值，则可能会收到此错误消息。布尔值必须为 TRUE 或 FALSE。确保您的条件表达式可以转换为布尔值。如果结果不是布尔值，则它等同于 FALSE 且不会调用操作或过渡到事件中指定的 nextState。

- 消息：在以下表达式*reference*中找到了不兼容的数据类型 [*inferred-types*]：*expression*

  解决方案：解决方案：探测器模型中所有具有相同输入属性或变量的表达式都必须引用相同的数据类型。

使用以下信息解决此问题：

- 当您将引用值作为一个或多个运算符的操作数时，确保您引用的所有数据类型均兼容。

  例如，在以下表达式中，整数 2 是 == 和 && 运算符的操作数。为确保操作数兼容，$variable.testVariable + 1 和 $variable.testVariable必须引用整数或小数。

  此外，整数 1 是运算符+的操作数。因此，$variable.testVariable 必须引用整数或小数。

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

- 当您使用引用作为传递给函数的自变量时，请确保该函数支持您引用的数据类型。

  例如，以下 timeout("*time-name*") 函数需要带有双引号的字符串作为自变量。如果使用引用作为*timer-name*值，则必须使用双引号引用字符串。

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

  > **Note**
  >
  > 对于convert(*type*, *expression*)函数，如果您使用引用作为*type*值，则引用的计算结果必须是StringDecimal、或Boolean。

  有关更多信息，请参阅 AWS IoT Events 表达式中输入和变量的参考。

- 消息：与使用的数据类型 [*inferred-types*] 不兼容*reference*。这可能会导致运行时系统错误。

  解决方案：如果同一个输入属性或变量的两个表达式引用了两种数据类型，则可能会收到此警告消息。确保同一输入属性或变量的表达式在探测器模型中引用相同的数据类型。

- 消息：您为运算符 [*inferred-types*] 输入的数据类型 [*operator*] 与以下表达式不兼容：'*expression*'

  解决方案：如果您的表达式组合了与指定运算符不兼容的数据类型，则可能会收到此错误消息。例如，在以下表达式中，运算符 + 与整数、小数值和字符串数据类型兼容，但不兼容布尔数据类型的操作数。

  ```
  true + false
  ```

  必须确保与运算符一起使用的数据类型兼容。

- 消息：找到的数据类型 [*inferred-types*] *input-attribute* 不兼容，可能导致运行时错误。

  解决方案：如果同一个输入属性的两个表达式引用了两种数据类型（要么是状态的 `OnEnterLifecycle`，要么是状态的 `OnInputLifecycle` 和 `OnExitLifecycle`），则可能会收到此错误消息。确保`OnEnterLifecycle`（或 `OnInputLifecycle` 和 `OnExitLifecycle`）中的表达式对探测器模型的每种状态都引用相同的数据类型。

- 消息：有效负载表达式 [*expression*] 无效。指定一个在运行时系统计算结果为字符串的表达式，因为负载类型为 JSON 格式。

  解决方案：如果您指定的负载类型为 JSON，但 AWS IoT Events 无法将其表达式计算为字符串，则可能会收到此错误。确保计算结果是字符串，而不是布尔值或数字。

- 消息：您的插值表达式 {*interpolated-expression*} 在运行时必须计算为整数或布尔值。否则，您的负载表达式 {*payload-expression*} 在运行时将无法解析为有效的 JSON。

  解决方案：如果 AWS IoT Events 无法将插值表达式计算为整数或布尔值，则可能会收到此错误消息。请确保插值表达式可以转换为整数或布尔值，因为不支持其他数据类型，例如字符串。

- 消息：`IotSitewiseAction`字段中的表达式类型定义*expression*为类型*defined-type*，推断为类型*inferred-type*。定义的类型和推断的类型必须相同。

  解决方案：如果 `IotSitewiseAction` 中的 `propertyValue` 表达式的数据类型定义与 AWS IoT Events推断的数据类型不同，则可能会收到此错误消息。确保在探测器模型中对该表达式的所有实例使用相同的数据类型。

- 消息：对于以下表达式，用于`setTimer`操作的数据类型 [*inferred-types*] `Integer` 的计算结果不是：*expression*

  解决方案：如果持续时间表达式的推断数据类型不是整数或小数值，则可能会收到此错误消息。确保 `durationExpression` 可转换为数字。不支持其他数据类型，如布尔值和字符串。

- 消息：与比较运算符 [*inferred-types*] 的操作数一起使用的数据类型 [*operator*] 在以下表达式中不兼容：*expression*

  解决方案：检测器模型的条件表达式 (*expression*) *operator* 中操作数的推断数据类型不匹配。操作数必须与探测器模型的所有其他部分中的匹配数据类型一起使用。

> **Tip**
>
> 您可以使用 `convert` 来更改探测器模型中表达式的数据类型。有关更多信息，请参阅 要在 AWS IoT Events 表达式中使用的函数。

referenced-data

包含 referenced-data 相关信息的分析结果对应以下错误消息：

- 消息：检测到损坏的计时器：计时器*timer-name*用于表达式中，但从未设置。

  解决方案：如果您使用未设置的计时器，则可能会收到此错误消息。在表达式中使用计时器之前，必须先设置计时器。另外，请确保输入正确的计时器名称。

- 消息：检测到损坏的变量：*variable-name*在表达式中使用了变量，但从未设置过变量。

  解决方案：如果您使用未设置的变量，则可能会收到此错误消息。在表达式中使用变量之前，必须先对变量进行设置。另外，请确保输入了正确的变量名称。

- 消息：检测到变量损坏：变量在设置为值之前在表达式中使用。

  解决方案：必须先为每个变量赋一个值，然后才能在表达式中对变量进行求值。每次使用前都要设置变量的值，以便可以检索其值。另外，请确保输入了正确的变量名称。

referenced-resource

包含 referenced-resource 相关信息的分析结果对应以下错误消息：

- 消息：探测器模型定义包含对不存在的输入的引用。

  解决方案：如果您使用表达式来引用不存在的输入，则可能会收到此错误消息。确保您的表达式引用现有的输入并输入正确的输入名称。如果没有输入，请创建一个输入。

- 消息：探测器模型定义包含无效 InputName：*input-name*

  解决方案：如果您的探测器模型包含无效的输入名称，则可能会收到此错误消息。确保输入了正确的输入名称。输入名称必须具有 1-128 个字符。有效字符：a-z、A-Z、0-9、_（下划线）和 -（连字符）。

分析探测器模型 AWS IoT Events（控制台）

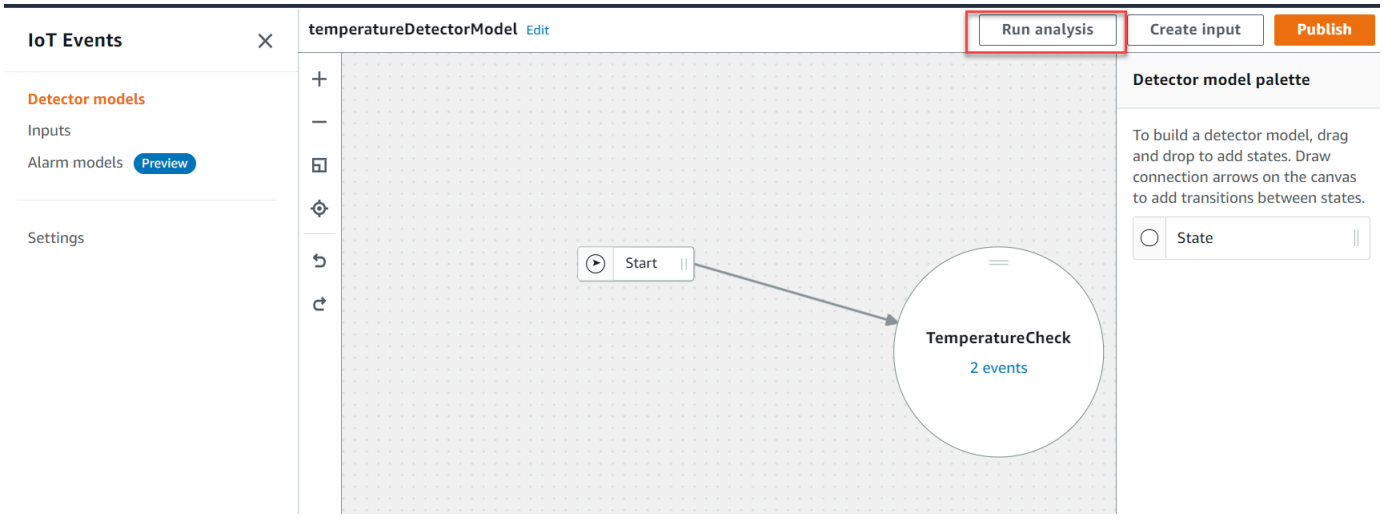
AWS IoT Events 允许您通过检测事件并使用 AWS IoT Events API 触发操作来监控物联网数据并做出反应。以下步骤使用 AWS IoT Events 控制台分析探测器模型。

> **Note**
>
> AWS IoT Events 开始分析您的探测器模型后，您最多有 24 小时的时间来检索分析结果。

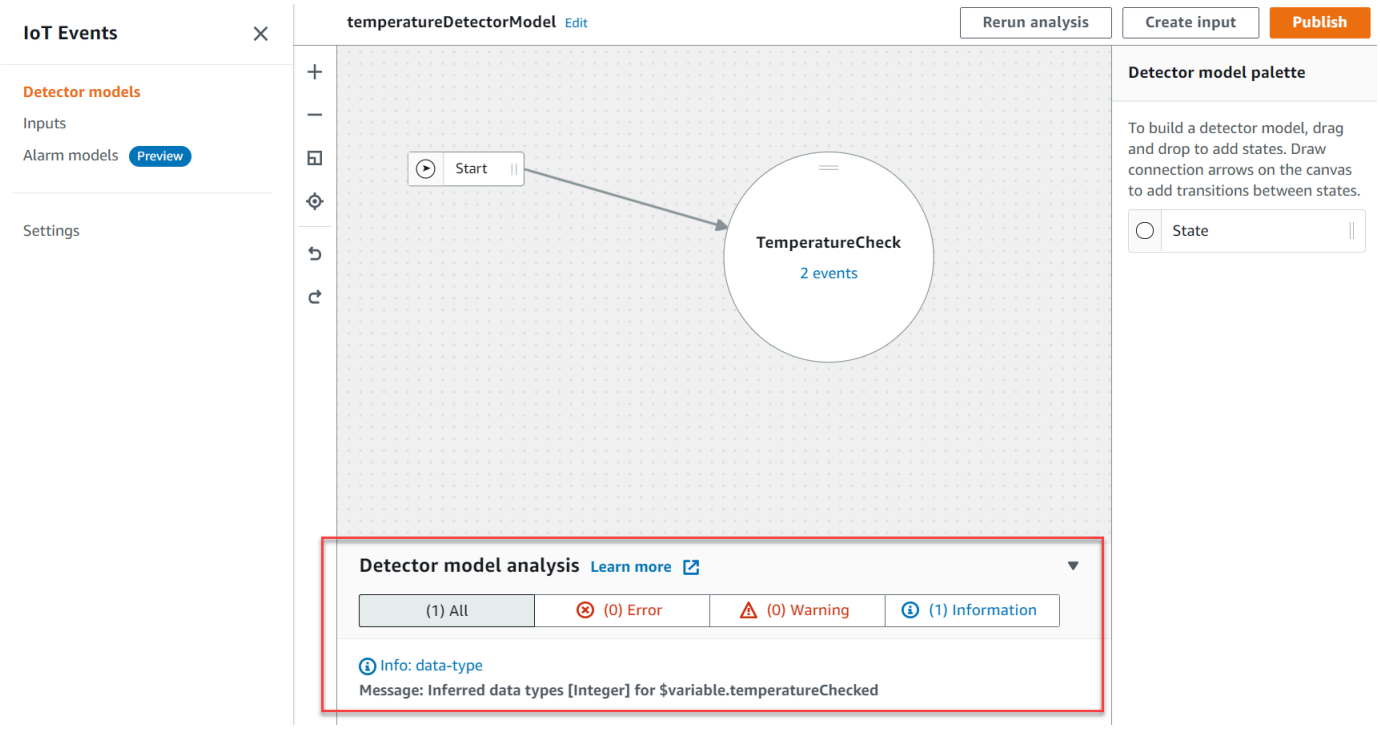
探测器模型分析可以帮助您优化模型、识别潜在问题并确保它们按预期运行。例如，在风电场上，探测器模型分析可以揭示模型是否根据异常振动模式正确识别潜在的齿轮故障。或者，如果模型在风速超过安全运行阈值时准确触发维护警报。通过根据分析完善模型，您可以改善预测性维护，减少停机时间并提高整体能源生产效率。

分析探测器模型

1. 登录 AWS IoT Events 控制台。
2. 在导航窗格中，选择探测器模型。
3. 在探测器模型下，选择目标探测器模型。
4. 在探测器模型页面上，选择编辑。
5. 在右上角，选择 运行分析。



以下是 AWS IoT Events 控制台中的示例分析结果。



在 AWS IoT Events (AWS CLI) 中分析探测器模型

以编程方式分析 AWS IoT Events 探测器模型，可提供有关其结构、行为和性能的宝贵见解。这种基于 API 的方法允许自动分析、与现有工作流程集成，并能够跨多个探测器模型执行批量操作。通过利用 StartDetectorModelAnalysisAPI，您可以启动对模型的深入检查，帮助您识别潜在问题，优化逻辑流程，并确保您的物联网事件处理符合您的业务需求。

以下步骤使用 AWS CLI 来分析探测器模型。

要使用分析探测器模型 AWS CLI

1. 运行以下命令以启动分析。

```
aws iotevents start-detector-model-analysis --cli-input-json file://file-name.json
```

> **Note**
>
> *file-name*替换为包含探测器模型定义的文件名。

Example探测器模型定义

```
{
    "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"
    }
}
```

如果您使用 AWS CLI 来分析现有的探测器模型，请选择以下选项之一来检索探测器模型定义：

- 如果要使用 AWS IoT Events 控制台，请执行以下操作：
  1. 在导航窗格中，选择探测器模型。
  2. 在探测器模型下，选择目标探测器模型。
  3. 从操作中选择导出探测器模型以下载探测器模型。此探测器模型以 JSON 格式保存。
  4. 打开探测器模型 JSON 文件。
  5. 您只需要 `detectorModelDefinition` 对象。移除以下内容：
     - 页面顶部的第一个大括号 ({)
     - `detectorModel` 线
     - `detectorModelConfiguration` 对象
     - 页面底部的最后一个大括号（}）
  6. 保存该文件。
- 如果要使用 AWS CLI，请执行以下操作：
  1. 在终端中运行以下命令。

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

  2. *detector-model-name*替换为探测器型号的名称。
  3. 将`detectorModelDefinition`对象复制至文本编辑器。
  4. 在`detectorModelDefinition`外面添加大括号 ({})。
  5. 以 JSON 格式保存此文件。

Example响应示例

```
{
    "analysisId": "c1133390-14e3-4204-9a66-31efd92a4fed"
}
```

2. 从输出项中复制分析 ID。
3. 运行以下命令，以检索分析状态。

```
aws iotevents describe-detector-model-analysis --analysis-id "analysis-id"
```

> **Note**
>
> *analysis-id*替换为您复制的分析 ID。

Example响应示例

```
{
    "status": "COMPLETE"
}
```

状态可以是以下值之一：

- RUNNING— AWS IoT Events 正在分析您的探测器模型。完成此过程可能最多需要一分钟。
- COMPLETE— AWS IoT Events 已完成对探测器模型的分析。
- FAILED— AWS IoT Events 无法分析您的探测器模型。请稍后重试。

4. 运行以下命令，以检索一个或多个探测器模型分析结果。

> **Note**
>
> *analysis-id*替换为您复制的分析 ID。

```
aws iotevents get-detector-model-analysis-results --analysis-id "analysis-id"
```

Example响应示例

```
{
    "analysisResults": [
        {
            "type": "data-type",
            "level": "INFO",
```

```
            "message": "Inferred data types [Integer] for
 $variable.temperatureChecked",
            "locations": []
        },
        {
            "type": "referenced-resource",
            "level": "ERROR",
            "message": "Detector Model Definition contains reference to Input
 'TemperatureInput' that does not exist.",
            "locations": [
                {
                    "path": "states[0].onInput.events[0]"
                }
            ]
        }
    ]
}
```

> **Note**
>
> AWS IoT Events 开始分析您的探测器模型后，您最多有 24 小时的时间来检索分析结果。

AWS IoT Events 命令

本章为中所有可用的 API 操作提供了全面的指南 AWS IoT Events。它提供了详细的解释，包括请求示例、响应以及支持的 Web 服务协议中每项操作的潜在错误。了解这些 API 操作有助于您有效地 AWS IoT Events 集成到物联网应用中，并自动执行事件检测和响应工作流程。

AWS IoT Events 行动

您可以使用 AWS IoT Events API 命令创建、读取、更新和删除输入和探测器模型，并列出其版本。有关更多信息，请参阅 AWS IoT Events API 参考 AWS IoT Events 中支持的操作和数据类型。

《AWS CLI 命令参考》中的AWS IoT Events 章节包括可用于管理和操作的 AWS CLI 命令 AWS IoT Events。

AWS IoT Events 数据

您可以使用 AWS IoT Events Data API 命令向探测器发送输入、列出探测器列表以及查看或更新探测器的状态。有关更多信息，请参阅 AWS IoT Events API 参考中的 D AWS IoT Events ata 支持的操作和数据类型。

《AWS CLI 命令参考》中的AWS IoT Events 数据部分包括可用于处理 AWS IoT Events 数据的 AWS CLI 命令。

的文档历史记录 AWS IoT Events

下表介绍了 2020 年 9 月 17 日之后对 AWS IoT Events 开发人员指南的重要更改。如需对此文档更新的通知，您可以订阅 RSS 源。

变更	说明	日期
终止支持通知	终止支持通知：2026 年 5 月 20 日， AWS 将停止对的支持。 AWS IoT Events 2026 年 5 月 20 日之后，您将无法再访问 AWS IoT Events 控制台或 AWS IoT Events 资源。	2025 年 5 月 20 日
区域启动	AWS IoT Events 现已在亚太地区（孟买）区域推出。	2021 年 9 月 30 日
区域发布	AWS IoT Events 现已在 AWS GovCloud （美国西部）地区推出。	2021 年 9 月 22 日
通过运行分析对探测器模型进行故障排除	AWS IoT Events 现在可以分析您的探测器模型并生成分析结果，用于对探测器模型进行故障排除。	2021 年 2 月 23 日
区域发布	AWS IoT Events 在中国（北京）推出。	2020 年 9 月 30 日
表达式用法	添加了演示如何编写表达式的示例。	2020 年 9 月 22 日
通过警报进行监控	警报可帮助您监控数据是否有变化。您可以创建警报，从而在突破阈值时发送通知。	2020 年 6 月 1 日

早期更新

下表介绍了 2020 年 9 月 18 日以前对AWS IoT Events 开发人员指南进行的一些重要更改。

更改	描述	日期
在表达式参考中添加了类型验证	在表达式参考中添加了类型验证信息。	2020 年 8 月 3 日
为其他服务添加了区域警告	添加了有关为和其他 AWS 服务选择相同区域 AWS IoT Events 的警告。	2020 年 5 月 7 日
添加、更新	• 有效载荷自定义功能 • 新的事件操作：亚马逊 DynamoDB 和 AWS IoT SiteWise	2020 年 4 月 27 日
为探测器模型条件表达式添加了内置函数	为探测器模型条件表达式添加了内置函数。	2019 年 9 月 10 日
添加了探测器模型示例	添加了探测器模型的示例。	2019 年 8 月 5 日
添加了新的事件操作	为以下内容添加了新的事件操作： • Lambda • Amazon SQS • Kinesis Data Firehose • AWS IoT Events 输入	2019 年 7 月 19 日
添加、更正	• 更新了`timeout()` 函数的描述。 • 添加了有关账户处于非活动状态的最佳实践。	2019 年 6 月 11 日
更新了权限策略和控制台调试选项	• 更新了控制台权限策略。	2019 年 6 月 5 日

更改	描述	日期
	• 更新了控制台调试选项页面图片。	
更新	AWS IoT Events 服务已全面开放。	2019 年 5 月 30 日
添加、更新	• 更新了安全信息。 • 添加了带注释的探测器模型示例。	2019 年 5 月 22 日
添加了示例和所需权限	添加了 Amazon SNS 有效负载示例；增加了所需的权限。`CreateDetectorModel`	2019 年 5 月 17 日
添加了其他安全信息	向安全部分添加了信息。	2019 年 5 月 9 日
有限预览版	文档的有限预览版。	2019 年 3 月 28 日