開發人員指南

AWS IoT Events

Copyright © 2026 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

AWS IoT Events: 開發人員指南

Copyright © 2026 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon 的商標和商業外觀不得用於任何非 Amazon 的產品或服務，也不能以任何可能造成客戶混淆、任何貶低或使 Amazon 名譽受損的方式使用 Amazon 的商標和商業外觀。所有其他非 Amazon 擁有的商標均為其各自擁有者的財產，這些擁有者可能附屬於 Amazon，或與 Amazon 有合作關係，亦或受到 Amazon 贊助。

Table of Contents

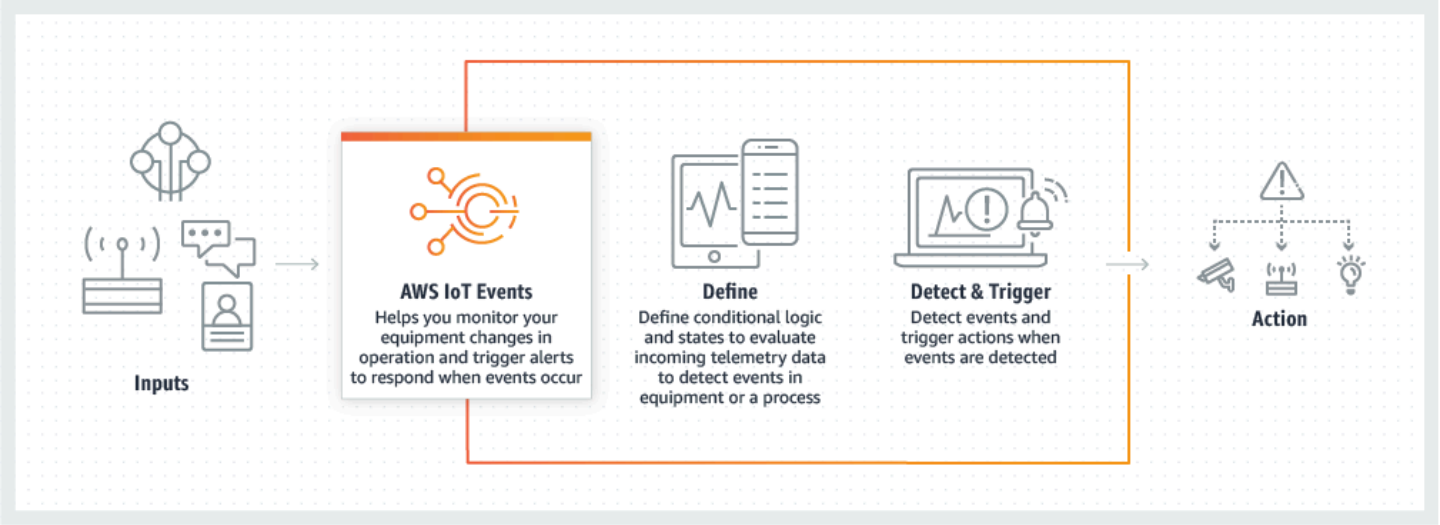
支援終止通知：2026 年 5 月 20 日， AWS 將終止對 的支援 AWS IoT Events。2026 年 5 月 20 日之後，您將無法再存取 AWS IoT Events 主控台或 AWS IoT Events 資源。如需詳細資訊，請參閱AWS IoT Events 終止支援。

本文為英文版的機器翻譯版本，如內容有任何歧義或不一致之處，概以英文版為準。

什麼是 AWS IoT Events？

AWS IoT Events 可讓您監控設備或裝置機群的運作失敗或變更，並在發生此類事件時觸發動作。AWS IoT Events 會持續監控來自裝置、程序、應用程式和其他 AWS 服務的 IoT 感應器資料，以識別重大事件，讓您可以採取動作。

使用 在 AWS 雲端中 AWS IoT Events 建置複雜的事件監控應用程式，您可以透過 AWS IoT Events 主控台或 APIs存取。



主題

優點和功能

接受來自多個來源的輸入

AWS IoT Events 接受來自許多 IoT 遙測資料來源的輸入。這些包括感應器裝置、管理應用程式和其他 AWS IoT 服務，例如 AWS IoT Core 和 AWS IoT Analytics。您可以使用標準 API 介面 AWS IoT Events (`BatchPutMessage` API) 或 AWS IoT Events 主控台，將任何遙測資料輸入推送至 。

如需開始使用 的詳細資訊 AWS IoT Events，請參閱AWS IoT Events 主控台入門。

使用簡單的邏輯表達式來識別複雜的事件模式

AWS IoT Events 可以識別涉及來自單一 IoT 裝置或應用程式的多個輸入，或來自各種設備和許多獨立感應器的事件模式。這特別有用，因為每個感應器和應用程式都提供重要資訊。但是，只有結合不同的感應器和應用程式資料，才能完整了解操作的效能和品質。您可以設定 AWS IoT Events 偵測器，使用簡單的邏輯表達式來識別這些事件，而不是複雜的程式碼。

如需邏輯表達式的詳細資訊，請參閱 篩選、轉換和處理事件資料的表達式。

根據事件觸發動作

AWS IoT Events 可讓您直接在 Amazon Simple Notification Service (Amazon SNS) AWS IoT Core、Lambda、Amazon SQS 和 Amazon Kinesis Firehose 中觸發動作。您也可以使用 AWS IoT 規則引擎來觸發 AWS Lambda 函數，這可讓您使用其他 服務，例如 Amazon Connect 或您自己的企業資源規劃 (ERP) 應用程式來採取動作。

AWS IoT Events 包含預先建置的動作程式庫，而且可讓您定義自己的動作。

若要進一步了解如何根據事件觸發動作，請參閱 在 中接收資料和觸發動作的支援動作 AWS IoT Events。

自動擴展以符合機群的需求

AWS IoT Events 當您連接同質裝置時， 會自動擴展。您可以為特定類型的裝置定義偵測器一次，服務會自動擴展和管理連線的裝置的所有執行個體 AWS IoT Events。

若要探索偵測器模型的範例，請參閱 AWS IoT Events 偵測器模型範例。

使用案例

AWS IoT Events 有許多用途。以下是幾個範例使用案例。

監控和維護遠端裝置

監控遠端部署的機器機群可能具有挑戰性，特別是當故障在沒有明確內容的情況下發生時。如果一台機器停止運作，這可能表示取代整個處理單元或機器。但這不是永續的。透過 AWS IoT Events ，您可以接收來自每部機器上多個感應器的訊息，以協助您診斷一段時間內的特定問題。您現在沒有取代整個單元，而是擁有必要的資訊，以傳送需要替換的確切部分給技術人員。使用數百萬台機器，節省最多可以增加數百萬美元，進而降低擁有或維護每部機器的總成本。

管理工業機器人

在設施中部署機器人以自動化套件移動，可大幅提升效率。為了將成本降至最低，機器人可以配備簡單、低成本的感應器，將資料回報給雲端。不過，有了數十個感應器和數百種操作模式，即時偵測問題可能具有挑戰性。使用 AWS IoT Events，您可以建置專家系統，在雲端處理此感應器資料，並建立提醒，以便在發生即將發生的故障時自動通知技術人員。

追蹤建置自動化系統

在資料中心，監控高溫和低濕度有助於防止設備故障。感應器通常從許多製造商購買，每種類型都隨附自己的管理軟體。不過，來自不同廠商的管理軟體有時不相容，因此難以偵測問題。使用 AWS IoT Events，您可以設定提醒，在發生故障之前，提前將加熱和冷卻系統的問題通知您的操作分析師。如此一來，您可以防止意外關閉資料中心，這將花費數千美元的設備替換成本，並可能損失收入。

AWS IoT Events 終止支援

在仔細考慮之後，我們決定終止 AWS IoT Events 對服務的支援，自 2026 年 5 月 20 日起。 AWS IoT Events 自 2025 年 5 月 20 日起將不再接受新客戶。身為在 2025 年 5 月 20 日之前註冊服務的現有客戶，您可以繼續使用 AWS IoT Events 功能。2026 年 5 月 20 日之後，您將無法再使用 AWS IoT Events。

此頁面提供客戶轉換至替代解決方案 AWS IoT Events 的說明和考量，以滿足您的業務需求。

> **Note**
>
> 這些指南中提供的解決方案旨在做為說明性範例，而不是做為 AWS IoT Events 功能的生產就緒替代。根據您的業務需求自訂程式碼、工作流程和相關 AWS 資源。

主題

從 遷移時的考量事項 AWS IoT Events

- 實作安全最佳實務，包括為每個元件使用具有最低權限的 IAM 角色，以及加密靜態和傳輸中的資料。如需詳細資訊，請參閱《IAM 使用者指南》中的 IAM 中的安全性最佳實務。
- 根據您的資料擷取需求，考慮 Kinesis 串流的碎片數量。如需 Kinesis 碎片的詳細資訊，請參閱《Amazon Kinesis Data Streams 開發人員指南》中的 Amazon Kinesis Data Streams 術語和概念。 Amazon Kinesis
- 使用 CloudWatch 為指標和日誌設定全面的監控和偵錯。如需詳細資訊，請參閱《Amazon CloudWatch 使用者指南》中的什麼是 CloudWatch？。 Amazon CloudWatch
- 考慮錯誤處理的結構，包括如何管理重複處理失敗的訊息、實作重試政策，以及設定程序來隔離和分析有問題的訊息。
- 使用 AWS 定價計算器來預估特定使用案例的成本。

中的偵測器模型遷移程序 AWS IoT Events

本節說明替代解決方案，可在您遷移時提供類似的偵測器模型功能 AWS IoT Events。

您可以透過 AWS IoT Core 規則將資料擷取遷移至其他服務 AWS 的組合。資料可以路由到 AWS IoT Core MQTT 主題，而不是透過 BatchPutMessage API 擷取資料。

此遷移方法利用 AWS IoT Core MQTT 主題做為 IoT 資料的進入點，取代 的直接輸入 AWS IoT Events。MQTT 主題的選擇有幾個主要原因。由於 MQTT 在業界的廣泛使用，它們提供了與 IoT 裝置的廣泛相容性。這些主題可以處理來自許多裝置的大量訊息，以確保可擴展性。它們也提供根據內容或裝置類型路由和篩選訊息的彈性。此外， AWS IoT Core MQTT 主題與其他 AWS 服務無縫整合，促進遷移程序。

資料從 MQTT 主題流向結合 Amazon Kinesis Data Streams、 AWS Lambda 函數、Amazon DynamoDB 資料表和 Amazon EventBridge 排程的架構。這種服務組合會複寫並增強 先前提供的功能 AWS IoT Events，讓您更靈活地控制 IoT 資料處理管道。

比較架構

目前的 AWS IoT Events 架構會透過 AWS IoT Core 規則和 `BatchPutMessage` API 擷取資料。此架構使用 AWS IoT Core 進行資料擷取和事件發佈，並將訊息透過 AWS IoT Events 輸入路由至定義狀態邏輯的偵測器模型。IAM 角色會管理必要的許可。

新的解決方案 AWS IoT Core 會維護資料擷取（現在具有專用輸入和輸出 MQTT 主題）。它引入用於資料分割的 Kinesis Data Streams 和用於狀態邏輯的評估器 Lambda 函數。裝置狀態現在存放在 DynamoDB 資料表中，增強型 IAM 角色會管理這些服務的許可。

用途	解決方案	差異
資料擷取 – 從 IoT 裝置接收資料	AWS IoT Core	現在需要兩個不同的 MQTT 主題：一個用於擷取裝置資料，另一個用於發佈輸出事件
訊息方向 – 將傳入的訊息路由到適當的服務	AWS IoT Core 訊息路由規則	維持相同的路由功能，但現在會將訊息導向 Kinesis Data Streams，而不是 AWS IoT Events
資料處理 – 處理和組織傳入的資料串流	Kinesis Data Streams	取代 AWS IoT Events 輸入功能，使用裝置 ID 分割提供資料擷取以進行訊息處理

用途	解決方案	差異
邏輯評估 – 處理狀態變更並觸發動作	評估器 Lambda	取代 AWS IoT Events 偵測器模型，透過程式碼而非視覺化工作流程提供可自訂的狀態邏輯評估
狀態管理 – 維護裝置狀態	DynamoDB 表	提供裝置狀態持久性儲存的新元件，取代內部 AWS IoT Events 狀態管理
安全性 – 管理服務許可	IAM 角色	已更新的許可現在除了現有 AWS IoT Core 許可之外，還包括對 Kinesis Data Streams、DynamoDB 和 EventBridge 的存取

步驟 1：（選用）匯出 AWS IoT Events 偵測器模型組態

建立新資源之前，請先匯出 AWS IoT Events 偵測器模型定義。這些包含您的事件處理邏輯，可以做為實作新解決方案的歷史參考。

Console

使用 匯出偵測器模型組態時 AWS IoT Events AWS 管理主控台，請執行下列步驟：

使用 匯出偵測器模型 AWS 管理主控台

1. 登入 AWS IoT Events 主控台。
2. 在左側導覽窗格中，選擇 Detector models (偵測器模型)。
3. 選取要匯出的偵測器模型。
4. 選擇 Export (匯出)。閱讀有關輸出的資訊訊息，然後再次選擇匯出。
5. 針對您要匯出的每個偵測器模型重複此程序。

包含偵測器模型 JSON 輸出的檔案會新增至瀏覽器的下載資料夾。您可以選擇性地儲存每個偵測器模型組態，以保留歷史資料。

AWS CLI

使用 AWS CLI執行下列命令來匯出偵測器模型組態：

使用 匯出偵測器模型 AWS CLI

1. 列出您帳戶中的所有偵測器模型：

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

2. 針對每個偵測器模型，執行下列動作來匯出其組態：

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

3. 儲存每個偵測器模型的輸出。

步驟 2：建立 IAM 角色

建立 IAM 角色以提供複寫 功能的許可 AWS IoT Events。此範例中的角色會授予 DynamoDB 的存取權以進行狀態管理、EventBridge 用於排程、Kinesis Data Streams 用於資料擷取、 AWS IoT Core 用於發佈訊息，以及 CloudWatch 用於記錄。這些服務共同用來取代 AWS IoT Events。

1. 建立具備下列許可的 IAM 角色。如需建立 IAM 角色的詳細說明，請參閱《IAM 使用者指南》中的建立角色以將許可委派給 AWS 服務。

   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. 新增下列 IAM 角色信任政策。信任政策允許指定的 AWS 服務擔任 IAM 角色，以便他們可以執行必要的動作。如需建立 IAM 信任政策的詳細說明，請參閱《IAM 使用者指南》中的使用自訂信任政策建立角色。

   JSON

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

步驟 3：建立 Amazon Kinesis Data Streams

使用 AWS 管理主控台 或 建立 Amazon Kinesis Data Streams AWS CLI。

Console

若要使用 建立 Kinesis 資料串流 AWS 管理主控台，請遵循 Amazon Kinesis Data Streams 開發人員指南中的建立資料串流頁面上的程序。

根據裝置計數和訊息承載大小調整碎片計數。

AWS CLI

使用 AWS CLI建立 Amazon Kinesis Data Streams，從您的裝置擷取和分割資料。

Kinesis Data Streams 用於此遷移，以取代 的資料擷取功能 AWS IoT Events。它提供可擴展且有效率的方式來從 IoT 裝置收集、處理和分析即時串流資料，同時提供靈活的資料處理和與其他 AWS 服務的整合。

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

根據裝置計數和訊息承載大小調整碎片計數。

步驟 4：建立或更新 MQTT 訊息路由規則

您可以建立新的 MQTT 訊息路由規則或更新現有的規則。

Console

1. 判斷您是否需要新的 MQTT 訊息路由規則，或是是否可以更新現有的規則。
2. 開啟 AWS IoT Core 主控台。
3. 在導覽窗格中，選擇訊息路由，然後選擇規則。
4. 在管理區段中，選擇訊息路由，然後選擇規則。
5. 選擇建立規則。
6. 在指定規則屬性頁面上，輸入 AWS IoT Core 規則名稱的規則名稱。針對規則描述 - 選用，輸入描述以識別您正在處理事件並將其轉送至 Kinesis Data Streams。
7. 在設定 SQL 陳述式頁面上，輸入 SQL 陳述式的下列項目：**`SELECT * FROM 'your-database'`**，然後選擇下一步。
8. 在連接規則動作頁面上的規則動作下，選擇 kinesis。
9. 選擇串流的 Kinesis 串流。針對分割區索引鍵，輸入 **`your-instance-id`**。選取 IAM 角色的適當角色，然後選擇新增規則動作。

如需詳細資訊，請參閱建立 AWS IoT 規則以將裝置資料路由至其他 服務。

AWS CLI

1. 使用下列內容建立 JSON 檔案 。此 JSON 組態檔案會定義 AWS IoT Core 規則，從主題中選取所有訊息，並使用執行個體 ID 做為分割區索引鍵，將其轉送至指定的 Kinesis 串流。

```
{
    "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 建立 MQTT 主題規則 AWS CLI。此步驟使用 AWS CLI ，使用 `events_rule.json` 檔案中定義的組態來建立 AWS IoT Core 主題規則。

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

步驟 5：取得目的地 MQTT 主題的端點

使用目的地 MQTT 主題來設定主題發佈傳出訊息的位置，取代先前處理的功能 AWS IoT Events。端點對 AWS 您的帳戶和區域是唯一的。

Console

1. 開啟 AWS IoT Core 主控台。
2. 在左側導覽面板的連線區段中，選擇網域組態。
3. 選擇 iot：Data-ATS 網域組態，以開啟組態的詳細資訊頁面。
4. 複製網域名稱值。此值是端點。儲存端點值，因為您會在後續步驟中需要它。

AWS CLI

執行下列命令以取得 AWS IoT Core 端點，以發佈您帳戶的傳出訊息。

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

步驟 6：建立 Amazon DynamoDB 資料表

Amazon DynamoDB 資料表會取代 的狀態管理功能 AWS IoT Events，提供可擴展且彈性的方式，在新的解決方案架構中保留和管理裝置狀態和偵測器模型邏輯。

Console

建立 Amazon DynamoDB 資料表以保留偵測器模型的狀態。如需詳細資訊，請參閱《Amazon DynamoDB 開發人員指南》中的在 DynamoDB 中建立資料表。 DynamoDB

使用下列表格詳細資訊：

- 針對資料表名稱，輸入您選擇的資料表名稱。
- 針對分割區索引鍵，輸入您自己的執行個體 ID。
- 您可以使用資料表設定的預設設定

AWS CLI

執行下列命令來建立 DynamoDB 資料表。

```
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 \
```

步驟 7：建立 AWS Lambda 函數 （主控台）

Lambda 函數做為核心處理引擎，取代 的偵測器模型評估邏輯 AWS IoT Events。在此範例中，我們與其他 AWS 服務整合，以根據您定義的規則處理傳入資料、管理狀態和觸發動作。

建立具有NodeJS執行時間的 Lambda 函數。使用下列程式碼片段，取代硬式編碼常數：

1. 開啟 AWS Lambda console。
2. 選擇建立函數。
3. 輸入函數名稱的名稱。
4. 選取 NodeJS 22.x 作為執行時間。

5. 在變更預設執行角色下拉式清單中，選擇使用現有角色，然後選取您在先前步驟中建立的 IAM 角色。
6. 選擇建立函數。
7. 在取代硬式編碼常數後，貼上下列程式碼片段。
8. 函數建立之後，請在程式碼索引標籤下貼上下列程式碼範例，以您自己的程式碼取代**your-destination-endpoint**端點。

```
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
        ]
    );
```

步驟 8：新增 Amazon Kinesis Data Streams 觸發條件

使用 AWS 管理主控台 或 將 Kinesis Data Streams 觸發新增至 Lambda 函數 AWS CLI。

將 Kinesis Data Streams 觸發條件新增至 Lambda 函數會建立資料擷取管道與處理邏輯之間的連線，讓它自動評估傳入的 IoT 資料串流，並即時對事件做出反應，類似於 AWS IoT Events 處理輸入的方式。

Console

如需詳細資訊，請參閱《 AWS Lambda 開發人員指南》中的建立事件來源映射以叫用 Lambda 函數。

針對事件來源映射詳細資訊，請使用下列項目：

- 針對函數名稱，輸入 中使用的 lambda 名稱步驟 7：建立 AWS Lambda 函數 （主控台）。
- 針對消費者 - 選用，輸入 Kinesis 串流的 ARN。
- 批次大小請輸入 **10**。

AWS CLI

執行下列命令來建立 Lambda 函數觸發。

```
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
```

步驟 9：測試資料擷取和輸出功能 (AWS CLI)

根據您在偵測器模型中定義的內容，將承載發佈至 MQTT 主題。以下是 MQTT 主題的範例承載，`your-topic-name`用於測試實作。

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

您應該會看到發佈至主題的 MQTT 訊息，其中包含下列 （或類似內容）：

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

中的 AWS IoT SiteWise 警示遷移程序 AWS IoT Events

本節說明替代解決方案，可在您遷移時提供類似的警示功能 AWS IoT Events。

對於使用 AWS IoT Events 警示的 AWS IoT SiteWise 屬性，您可以使用 CloudWatch 警示遷移至解決方案。此方法透過已建立的 SLAs 和其他功能提供強大的監控功能，例如異常偵測和分組警示。

比較架構

屬性目前的 AWS IoT Events 警示組態 AWS IoT SiteWise 需要在資產模型`AssetModelCompositeModels`中建立 ，如AWS IoT SiteWise 《 使用者指南》中的在 中定義外部警示 AWS IoT SiteWise中所述。對新解決方案的修改通常透過 主控台進行 AWS IoT Events 管理。

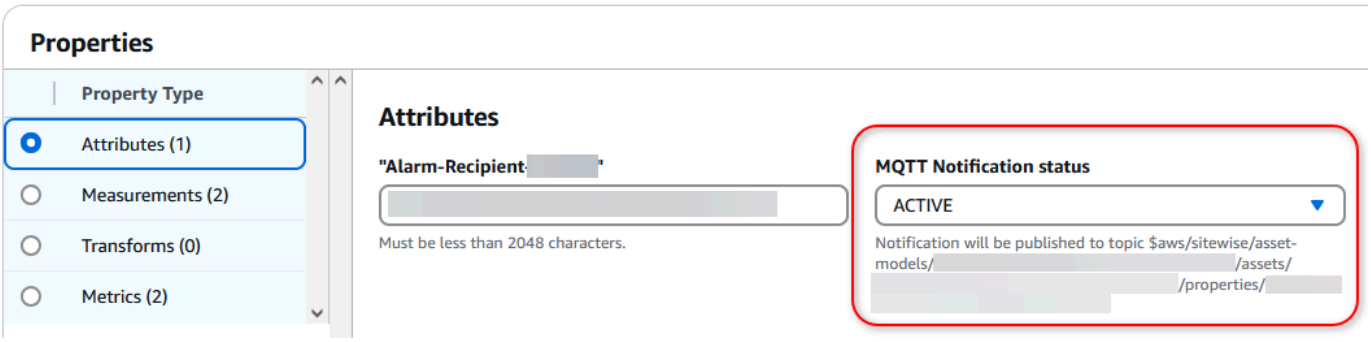
新解決方案利用 CloudWatch 警示提供警示管理。此方法使用 AWS IoT SiteWise 通知將屬性資料點發佈至 AWS IoT Core MQTT 主題，然後由 Lambda 函數處理。函數會將這些通知轉換為 CloudWatch 指標，透過 CloudWatch 的警示架構啟用警示監控。

用途	解決方案	差異
資料來源 – 來自 的屬性資料 AWS IoT SiteWise	AWS IoT SiteWise MQTT 通知	使用 AWS IoT SiteWise 屬性的 MQTT 通知取代直接 IoT Events 整合
資料處理 – 轉換屬性資料	Lambda 函式	處理 AWS IoT SiteWise 屬性通知並將其轉換為 CloudWatch 指標
警示評估 – 監控指標並觸發警示	Amazon CloudWatch 警示	以 CloudWatch AWS IoT Events 警示取代警示，提供異常偵測等其他功能
整合 – 與 的連線 AWS IoT SiteWise	AWS IoT SiteWise 外部警示	將 CloudWatch 警示匯入回 AWS IoT SiteWise 做為外部警示的選用功能

步驟 1：在資產屬性上啟用 MQTT 通知

如果您使用 AWS IoT SiteWise 警示的 AWS IoT Events 整合，您可以為要監控的每個屬性開啟 MQTT 通知。

1. 遵循程序中的資產設定警示 AWS IoT SiteWise，直到您完成編輯資產模型屬性的每個步驟。
2. 對於每個要遷移的屬性，請將 MQTT 通知狀態變更為 ACTIVE。



3. 請注意警示針對每個修改的警示屬性發佈的主題路徑。

如需詳細資訊，請參閱下列文件資源：

- AWS IoT SiteWise 《使用者指南》中的了解 MQTT 主題中的資產屬性。
- 《AWS IoT 開發人員指南》中的 MQTT 主題。

步驟 2：建立 AWS Lambda 函數

建立 Lambda 函數以讀取 MQTT 主題發佈的 TQV 陣列，並將個別值發佈至 CloudWatch。我們將使用此 Lambda 函數作為在 AWS IoT Core 訊息規則中觸發的目的地動作。

1. 開啟 AWS Lambda console。
2. 選擇建立函數。
3. 輸入函數名稱的名稱。
4. 選取 NodeJS 22.x 做為執行時間。
5. 在變更預設執行角色下拉式清單中，選擇使用現有角色，然後選取您在先前步驟中建立的 IAM 角色。

> **Note**
>
> 此程序假設您已遷移偵測器模型。如果您沒有 IAM 角色，請參閱 ???。

6. 選擇建立函數。
7. 在取代硬式編碼常數後，貼上下列程式碼片段。

```
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)}')
        }
```

步驟 3：建立 AWS IoT Core 訊息路由規則

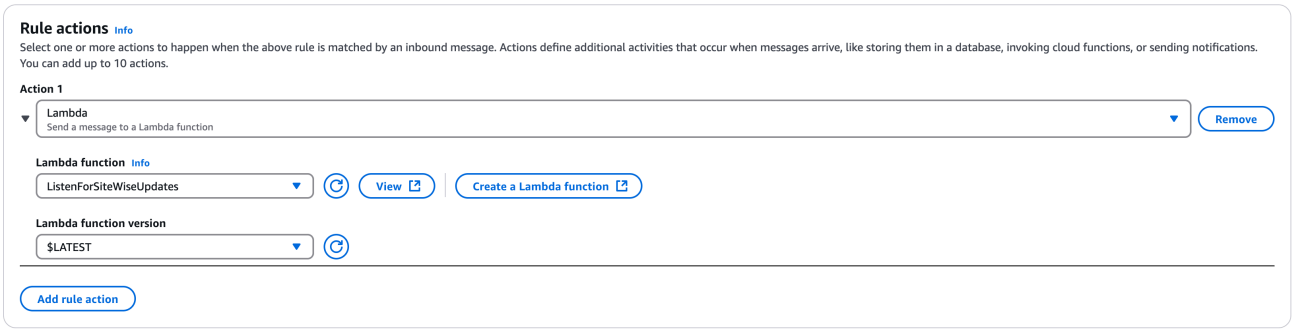
- 遵循教學課程：重新發佈 MQTT 訊息程序，在出現提示時輸入下列資訊：

  a. 命名訊息路由規則 `SiteWiseToCloudwatchAlarms`。

  b. 對於查詢，您可以使用下列項目：

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

  c. 在規則動作中，選取 Lambda 動作，將產生的資料 AWS IoT SiteWise 從 傳送至 CloudWatch。例如：



步驟 4：檢視 CloudWatch 指標

當您將資料擷取至 中稍早選取的 AWS IoT SiteWise屬性步驟 1：在資產屬性上啟用 MQTT 通知時，會將資料路由至我們在 中建立的 Lambda 函數步驟 2：建立 AWS Lambda 函數。在此步驟中，您可以檢查 以查看將指標傳送至 CloudWatch 的 Lambda。

1. 開啟 CloudWatch AWS 管理主控台。
2. 在左側導覽中，選擇指標，然後選擇所有指標。

3. 選擇警示的 URL 來開啟它。
4. 在來源索引標籤下，CloudWatch 輸出看起來與此範例類似。此來源資訊會確認指標資料正在饋送至 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"
}
```

步驟 5：建立 CloudWatch 警示

遵循《Amazon CloudWatch 使用者指南》中的根據靜態閾值程序建立 CloudWatch 警示，為每個相關指標建立警示。 Amazon CloudWatch

> **Note**
>
> Amazon CloudWatch 中有許多警示組態選項。如需 CloudWatch 警示的詳細資訊，請參閱Amazon CloudWatch 使用者指南》中的使用 Amazon CloudWatch 警示。 Amazon CloudWatch

步驟 6：（選用）將 CloudWatch 警示匯入至 AWS IoT SiteWise

您可以設定 CloudWatch 警示， AWS IoT SiteWise 使用 CloudWatch 警示動作和 Lambda 將資料傳回至 。此整合可讓您在 SiteWise Monitor 入口網站中檢視警示狀態和屬性。

1. 將外部警示設定為資產模型中的屬性。如需詳細資訊，請參閱AWS IoT SiteWise 《使用者指南》中的在 中定義外部警示 AWS IoT SiteWise。
2. 建立 Lambda 函數，使用AWS IoT SiteWise 《使用者指南》中的 BatchPutAssetPropertyValue API 將警示資料傳送至 AWS IoT SiteWise。
3. 設定 CloudWatch 警示動作，以在警示狀態變更時叫用 Lambda 函數。如需詳細資訊，請參閱《Amazon CloudWatch 使用者指南》中的警示動作一節。

設定 AWS IoT Events

本節提供設定 的指南 AWS IoT Events，包括建立 AWS 帳戶、設定必要的許可，以及建立管理 資源存取的角色。

主題

設定 AWS 帳戶

註冊 AWS 帳戶

如果您沒有 AWS 帳戶，請完成下列步驟來建立一個。

註冊 AWS 帳戶

1. 開啟 https://portal.aws.amazon.com/billing/signup。
2. 請遵循線上指示進行。

   部分註冊程序需接收來電或簡訊，並在電話鍵盤輸入驗證碼。

   當您註冊 時 AWS 帳戶，AWS 帳戶根使用者會建立 。根使用者有權存取該帳戶中的所有 AWS 服務 和資源。作為安全最佳實務，請將管理存取權指派給使用者，並且僅使用根使用者來執行需要根使用者存取權的任務。

AWS 會在註冊程序完成後傳送確認電子郵件給您。您可以隨時登錄 https://aws.amazon.com/ 並選擇我的帳戶，以檢視您目前的帳戶活動並管理帳戶。

建立具有管理存取權的使用者

註冊 後 AWS 帳戶，請保護 AWS 帳戶根使用者、啟用 AWS IAM Identity Center和建立管理使用者，以免將根使用者用於日常任務。

保護您的 AWS 帳戶根使用者

1. 選擇根使用者並輸入 AWS 帳戶 您的電子郵件地址，以帳戶擁有者AWS 管理主控台身分登入 。在下一頁中，輸入您的密碼。

如需使用根使用者登入的說明，請參閱 AWS 登入 使用者指南中的以根使用者身分登入。

2. 若要在您的根使用者帳戶上啟用多重要素驗證 (MFA)。

   如需說明，請參閱《IAM 使用者指南》中的為您的 AWS 帳戶 根使用者 （主控台） 啟用虛擬 MFA 裝置。

建立具有管理存取權的使用者

1. 啟用 IAM Identity Center。

   如需指示，請參閱《AWS IAM Identity Center 使用者指南》中的啟用 AWS IAM Identity Center。

2. 在 IAM Identity Center 中，將管理存取權授予使用者。

   如需使用 IAM Identity Center 目錄 做為身分來源的教學課程，請參閱AWS IAM Identity Center 《 使用者指南》中的使用預設值設定使用者存取 IAM Identity Center 目錄。

以具有管理存取權的使用者身分登入

- 若要使用您的 IAM Identity Center 使用者簽署，請使用建立 IAM Identity Center 使用者時傳送至您電子郵件地址的簽署 URL。

  如需使用 IAM Identity Center 使用者登入的說明，請參閱AWS 登入 《 使用者指南》中的登入 AWS 存取入口網站。

指派存取權給其他使用者

1. 在 IAM Identity Center 中，建立一個許可集來遵循套用最低權限的最佳實務。

   如需指示，請參閱《AWS IAM Identity Center 使用者指南》中的建立許可集。

2. 將使用者指派至群組，然後對該群組指派單一登入存取權。

   如需指示，請參閱《AWS IAM Identity Center 使用者指南》中的新增群組。

設定 的許可 AWS IoT Events

實作適當的許可對於安全且有效地使用 至關重要 AWS IoT Events。本節說明使用 的某些功能所需的許可 AWS IoT Events。您可以使用 AWS CLI 命令或 AWS Identity and Access Management (IAM) 主控台來建立角色和相關聯的許可政策，以存取 資源或執行特定 函數 AWS IoT Events。

IAM 使用者指南提供有關安全控制 AWS 資源存取許可的詳細資訊。如需 的特定資訊 AWS IoT Events，請參閱 的動作、資源和條件索引鍵 AWS IoT Events。

若要使用 IAM 主控台建立和管理角色和許可，請參閱 IAM 教學課程：使用 IAM 角色在 AWS 帳戶之間委派存取權。

> **Note**
>
> 金鑰可以是 1-128 個字元，可包含：
>
> - 大小寫字母 a-z
> - 數字 0-9
> - 特殊字元 -、_ 或 ：。

的動作許可 AWS IoT Events

AWS IoT Events 可讓您觸發使用其他 AWS 服務的動作。若要這樣做，您必須授予代表您執行這些動作的 AWS IoT Events 許可。本節包含動作清單和範例政策，該政策會授予在您的 資源上執行所有這些動作的許可。視需要變更*##*和*## ID* 參考。如果可能，您也應該變更萬用字元 (*)，以參考將存取的特定資源。您可以使用 IAM 主控台授予 許可， AWS IoT Events 以傳送您定義的 Amazon SNS 提醒。

AWS IoT Events 支援下列動作，可讓您使用計時器或設定變數：

- `setTimer` 來建立計時器。
- `resetTimer` 以重設計時器。
- `clearTimer` 刪除計時器。
- `setVariable` 來建立變數。

AWS IoT Events 支援下列動作，可讓您使用 AWS 服務：

- iotTopicPublish 在 MQTT 主題上發佈訊息。
- iotEvents 將資料傳送至 AWS IoT Events 做為輸入值。
- iotSiteWise 將資料傳送至 AWS IoT SiteWise中的資產屬性。
- dynamoDB 將資料傳送至 Amazon DynamoDB 資料表。
- dynamoDBv2 將資料傳送至 Amazon DynamoDB 資料表。
- firehose 將資料傳送至 Amazon Data Firehose 串流。
- lambda 叫用 AWS Lambda 函數。
- sns 以推送通知的形式傳送資料。
- sqs 將資料傳送至 Amazon SQS 佇列。

Example政策

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:*"
        }
    ]
}
```

保護 中的輸入資料 AWS IoT Events

請務必考慮誰可以授予輸入資料的存取權，以便在偵測器模型中使用。如果您有想要限制整體許可的使用者或實體，但允許建立或更新偵測器模型，您還必須授予該使用者或實體更新輸入路由的許可。這表示除了授予 `iotevents:CreateDetectorModel`和的許可`iotevents:UpdateDetectorModel`，您還必須授予 的許可`iotevents:UpdateInputRouting`。

Example

下列政策會新增 的許可`iotevents:UpdateInputRouting`。

JSON

```
{
```

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

您可以為 "*" 指定輸入 Amazon Resource Name (ARNs) 的清單，而不是萬用字元 "Resource"，以將此許可限制為特定輸入。這可讓您限制存取由使用者或實體建立或更新的偵測器模型所耗用的輸入資料。

的 Amazon CloudWatch 記錄角色政策 AWS IoT Events

下列政策文件提供角色政策和信任政策，允許 AWS IoT Events 代表您將日誌提交至 CloudWatch。

角色政策：

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:*:*:*"
```

```
            ]
        }
    ]
}
```

信任政策：

JSON

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

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

您也需要連接到使用者的 IAM 許可政策，以允許使用者傳遞角色，如下所示。如需詳細資訊，請參閱《IAM 使用者指南》中的授予使用者將角色傳遞至 AWS 服務的許可。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "",
            "Effect": "Allow",
            "Action": [
                "iam:GetRole",
                "iam:PassRole"
```

```
            ],
            "Resource": "arn:aws:iam::123456789012:role/Role_To_Pass"
        }
    ]
}
```

您可以使用下列命令來放置 CloudWatch 日誌的資源政策。這可讓 AWS IoT Events 將日誌事件放入 CloudWatch 串流。

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

使用下列命令來放置記錄選項。`roleArn` 將 取代為您建立的記錄角色。

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

的 Amazon SNS 訊息角色政策 AWS IoT Events

AWS IoT Events 與 Amazon SNS 整合需要謹慎的許可管理，才能提供安全且有效率的通知。本指南會逐步引導您設定 IAM 角色和政策，以允許 AWS IoT Events 將訊息發佈至 Amazon SNS 主題。

下列政策文件提供允許 傳送 SNS 訊息的角色政策和信任政策 AWS IoT Events 。

角色政策：

JSON

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

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

信任政策：

JSON

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

AWS IoT Events 主控台入門

本節說明如何使用 AWS IoT Events 主控台建立輸入和偵測器模型。您建立引擎的兩個狀態模型：正常狀態和過壓條件。當引擎中的測量壓力超過特定閾值時，模型會從正常狀態轉換為過壓狀態。然後，它會傳送 Amazon SNS 訊息來提醒技術人員有關 條件。當壓力再次低於連續三個壓力讀數的閾值時，模型會返回正常狀態，並傳送另一個 Amazon SNS 訊息作為確認。

我們會檢查是否有三個連續讀數低於壓力閾值，以消除在非線性復原階段或異常壓力讀數的情況下，可能發生的過壓或正常訊息停滯。

在 主控台上，您也可以找到幾個您可以自訂的預先製作偵測器模型範本。您也可以使用 主控台匯入其他人撰寫或匯出偵測器模型的偵測器模型，並在不同的 AWS 區域中使用它們。如果您匯入偵測器模型，請確定您為新區域建立所需的輸入或重新建立這些輸入，並更新使用的任何角色 ARNs。

使用 AWS IoT Events 主控台來了解以下內容。

定義輸入

若要監控您的裝置和程序，它們必須能夠將遙測資料匯入 AWS IoT Events。這可透過將訊息作為輸入傳送到 來完成 AWS IoT Events。您可以數種方式來執行此動作：

- 使用 BatchPutMessage 操作。
- 在 中 AWS IoT Core，為轉送訊息資料的 AWS IoT 規則引擎撰寫AWS IoT Events 動作規則 AWS IoT Events。您必須依名稱識別輸入。
- 在 中 AWS IoT Analytics，使用 CreateDataset 操作建立具有 的資料集`contentDeliveryRules`。這些規則會指定自動傳送資料集內容的 AWS IoT Events 輸入。

您必須先定義一或多個輸入，您的裝置才能以這種方式傳送資料。若要這樣做，請為每個輸入命名，並指定輸入監控器傳入訊息資料中的哪些欄位。

建立偵測器模型

使用 狀態定義偵測器模型 （設備或程序的模型）。針對每個狀態，定義評估傳入輸入的條件式（布林值）邏輯，以偵測重大事件。當偵測器模型偵測到事件時，它可以變更狀態，或使用其他 AWS 服務啟動自訂建置或預先定義的動作。您可以定義其他事件，在進入或退出狀態時啟動動作，也可以選擇在符合條件時啟動動作。

在本教學課程中，您會在模型進入或退出特定狀態時傳送 Amazon SNS 訊息做為動作。

監控裝置或程序

如果您監控多個裝置或程序，請在每個輸入中指定欄位，以識別輸入來自的特定裝置或程序。請參閱 中的 `key` 欄位`CreateDetectorModel`。當 識別的輸入欄位`key`辨識新值時，會識別新裝置並建立偵測器。每個偵測器都是偵測器模型的執行個體。新的偵測器會繼續回應來自該裝置的輸入，直到其偵測器模型更新或刪除為止。

如果您監控單一程序 （即使有多個裝置或子程序正在傳送輸入），則不會指定唯一的識別`key`欄位。在此情況下，模型會在第一個輸入到達時建立單一偵測器 （執行個體）。

將訊息做為輸入傳送到偵測器模型

有幾種方式可以從裝置或程序傳送訊息，做為 AWS IoT Events 偵測器的輸入，不需要您對訊息執行其他格式設定。在本教學課程中，您會使用 AWS IoT 主控台為轉送訊息資料的 AWS IoT 規則引擎撰寫AWS IoT Events 動作規則 AWS IoT Events。

若要執行此作業，請依名稱識別輸入，並繼續使用 AWS IoT 主控台來產生做為輸入轉送的訊息 AWS IoT Events。

> **Note**
>
> 本教學課程使用 主控台來建立相同的 `input`，並`detector model`顯示於 的範例AWS IoT Events 使用案例的教學課程。您可以使用此 JSON 範例來協助您遵循教學課程。

主題

開始使用 的先決條件 AWS IoT Events

如果您沒有 AWS 帳戶，請建立一個。

1. 遵循 中的步驟設定 AWS IoT Events，以確保適當的帳戶設定和許可。
2. 建立兩個 Amazon Simple Notification Service (Amazon SNS) 主題。

本教學課程（和對應的範例）假設您建立了兩個 Amazon SNS 主題。這些主題ARNs 會顯示為：`arn:aws:sns:us-east-1:123456789012:underPressureAction`和 `arn:aws:sns:us-east-1:123456789012:pressureClearedAction`。將這些值取代為您建立的 Amazon SNS 主題ARNs。如需詳細資訊，請參閱《Amazon Simple Notification Service 開發人員指南》。

除了將警示發佈到 Amazon SNS 主題外，您也可以讓偵測器傳送 MQTT 訊息，其中包含您指定的主題。使用此選項，您可以使用 AWS IoT Core 主控台訂閱和監控傳送至這些 MQTT 主題的訊息，來驗證偵測器模型是否正在建立執行個體，以及這些執行個體是否正在傳送提醒。您也可以使用在偵測器模型中建立的輸入或變數，在執行時間動態定義 MQTT 主題名稱。

3. 選擇 AWS 區域 支援的 AWS IoT Events。如需詳細資訊，請參閱 AWS 一般參考 中的 AWS IoT Events。如需協助，請參閱《 入門》中的 AWS 管理主控台服務入門 AWS 管理主控台。

在 中建立模型的輸入 AWS IoT Events

當您建構模型的輸入時，建議您收集檔案，其中包含裝置或程序傳送的範例訊息承載，以報告其運作狀態。擁有這些檔案可協助您定義必要的輸入。

您可以透過本節所述的多種方法建立輸入。

建立 JSON 輸入檔案

1. 若要開始使用，`input.json`請使用下列內容在本機檔案系統上建立名為 的檔案：

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

2. 現在您已擁有此入門`input.json`檔案，您可以建立 輸入。有兩種方法可以建立輸入。您可以使用AWS IoT Events 主控台 中的導覽窗格來建立輸入。或者，您可以在偵測器模型內建立建立輸入。

建立和設定輸入

了解如何為警示模型或偵測器模型建立輸入。

1. 登入 AWS IoT Events 主控台，或選取建立新 AWS IoT Events 帳戶的選項。
2. 在 AWS IoT Events 主控台的左上角，選取並展開導覽窗格。
3. 在左側導覽窗格中，選取輸入。
4. 在主控台的右上角，選擇建立輸入。
5. 提供 uniqueInputName。
6. 選用 – 輸入輸入的描述。
7. 若要上傳 JSON 檔案，請選取您在 概觀中建立`input.json`的檔案建立 JSON 輸入檔案。選擇輸入屬性，並顯示您輸入的屬性清單。
8. 針對選擇輸入屬性，選取要使用的屬性，然後選擇建立。在此範例中，我們選取 motorid 和 sensorData.pressure
9. 選用 – 將相關標籤新增至輸入。

> **Note**
>
> 您也可以在AWS IoT Events 主控台 的偵測器模型內建立其他輸入。如需詳細資訊，請參閱在的偵測器模型中建立輸入 AWS IoT Events。

在 的偵測器模型中建立輸入 AWS IoT Events

中的偵測器輸入 AWS IoT Events 做為資料來源和偵測器模型之間的橋樑。偵測器輸入提供支援 事件偵測和自動化功能的原始資料 AWS IoT Events。了解如何設定偵測器輸入，以協助模型準確回應 IoT 生態系統中的真實世界事件和條件。

本節說明如何定義偵測器模型的輸入，以接收遙測資料或訊息。

定義偵測器模型的輸入

1. 開啟 AWS IoT Events 主控台。
2. 在 AWS IoT Events 主控台中，選擇建立偵測器模型。
3. 選擇建立新項目。
4. 選擇 Create input (建立輸入)。
5. 針對輸入，輸入 InputName、選用的描述，然後選擇上傳檔案。在顯示的對話方塊中，選取您在概觀中建立`input.json`的檔案建立 JSON 輸入檔案。

6. 針對選擇輸入屬性，選取要使用的屬性，然後選擇建立。在此範例中，我們選取 motorId 和 sensorData.pressure

在 中建立偵測器模型 AWS IoT Events

在本主題中，您會使用 狀態定義偵測器模型 （設備或程序的模型）。

對於每個狀態，您定義條件式 （布林值） 邏輯，評估傳入輸入以偵測重大事件。偵測到事件時，它會變更狀態並可以啟動其他動作。這些事件稱為轉換事件。

在您的狀態中，您也可以定義每當偵測器進入或退出該狀態，或收到輸入 （這些稱為 `OnEnter`、`OnExit`和 `OnInput`事件） 時，可執行動作的事件。只有在事件的條件式邏輯評估為 時，才會執行動作`true`。

建立偵測器模型

1. 第一個偵測器狀態已為您建立。若要修改，請在主要編輯空間中選取帶有標籤 State_1 的圓圈。
2. 在狀態窗格中，輸入狀態名稱和 OnEnter，選擇新增事件。
3. 在新增 OnEnter 事件頁面上，輸入事件名稱和事件條件。在此範例中，輸入 `true`表示進入 狀態時一律會啟動事件。
4. 在事件動作下，選擇新增動作。
5. 在事件動作下，執行下列動作：

   a. 選取設定變數

   b. 針對變數操作，選擇指派值。

   c. 針對變數名稱，輸入要設定的變數名稱。

   d. 針對變數值，輸入值 **`0`**（零）。
6. 選擇儲存。

   如同您定義的變數，變數可以在偵測器模型的任何事件中設定 （提供值）。只有在偵測器達到 狀態並執行定義或設定的動作之後，才能參考變數的值 （例如，在事件的條件式邏輯中）。
7. 在狀態窗格中，選擇狀態旁邊的 X，以返回偵測器模型調色盤。
8. 若要建立第二個偵測器狀態，請在偵測器模型調色盤中，選擇狀態並將其拖曳至主要編輯空間。這會建立名為 的狀態`untitled_state_1`。
9. 在第一個狀態 (正常) 上暫停。狀態的圓周上會出現箭頭。

10. 按一下並將箭頭從第一個狀態拖曳至第二個狀態。從第一個狀態到第二個狀態（標記無標題）的導向線隨即出現。
11. 選取無標題行。在轉換事件窗格中，輸入事件名稱和事件觸發邏輯。
12. 在轉換事件窗格中，選擇新增動作。
13. 在新增轉換事件動作窗格中，選擇新增動作。
14. 針對選擇動作，選擇設定變數。
    a. 針對變數操作，選擇指派值。
    b. 針對變數名稱，輸入變數的名稱。
    c. 針對指派值，輸入值，例如：`$variable.pressureThresholdBreached + 3`
    d. 選擇儲存。
15. 選取第二個狀態 untitled_state_1。
16. 在狀態窗格中，輸入狀態名稱，然後在輸入時選擇新增事件。
17. 在新增 OnEnter 事件頁面上，輸入事件名稱和事件條件。選擇新增動作。
18. 針對選擇動作，選擇傳送 SNS 訊息。
    a. 針對 SNS 主題，輸入 Amazon SNS 主題的目標 ARN。
    b. 選擇儲存。
19. 繼續在範例中新增事件。
    a. 針對 OnInput，選擇新增事件，然後輸入並儲存下列事件資訊。

```
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. 針對 OnInput，選擇新增事件，然後輸入並儲存下列事件資訊。

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

```
      Variable operation: Decrement
      Variable name: pressureThresholdBreached
```

c. 針對 OnExit，選擇新增事件，然後使用您建立之 Amazon SNS 主題的 ARN 輸入並儲存下列事件資訊。

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

20. 在第二個狀態 (危險） 上暫停。狀態的圓周上會出現箭頭
21. 按一下並將箭頭從第二個狀態拖曳到第一個狀態。帶有標籤 Untitled 的導向行隨即出現。
22. 選擇無標題行，然後在轉換事件窗格中，使用下列資訊輸入事件名稱和事件觸發邏輯。

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

如需有關為什麼我們在觸發邏輯中測試 $input 值和 $variable值的詳細資訊，請參閱 中變數值可用性的項目AWS IoT Events 偵測器模型限制。

23. 選取啟動狀態。根據預設，此狀態會在您建立偵測器模型時建立）。在開始窗格中，選擇目的地狀態（例如，正常)。
24. 接著，將偵測器模型設定為接聽輸入。在右上角，選擇發佈。
25. 在發佈偵測器模型頁面上，執行下列動作。

a. 輸入偵測器模型名稱、描述和角色的名稱。此角色是為您建立的。

b. 為每個唯一的索引鍵值選擇建立偵測器。若要建立和使用您自己的角色，請遵循中的步驟，設定 的許可 AWS IoT Events並將其輸入為此處的角色。

26. 針對偵測器建立金鑰，選擇您先前定義之輸入其中一個屬性的名稱。您選擇做為偵測器建立金鑰的屬性必須存在於每個訊息輸入中，而且對於每個傳送訊息的裝置必須是唯一的。此範例使用 motorid 屬性。
27. 選擇 Save and Publish (儲存並發佈)。

> **Note**
>
> 為指定偵測器模型建立的唯一偵測器數量是根據傳送的輸入訊息。建立偵測器模型時，會從輸入屬性中選取金鑰。此金鑰決定要使用的偵測器執行個體。如果之前沒有看到金鑰（針對此偵測器模型），則會建立新的偵測器執行個體。如果之前已看到金鑰，我們會使用與此金鑰值對應的現有偵測器執行個體。

您可以重新建立或更新偵測器模型定義的備份副本（以 JSON 表示），或使用 作為範本來建立另一個偵測器模型。

您可以從 主控台或使用下列 CLI 命令來執行此操作。如有必要，請變更偵測器模型的名稱，以符合您在上一個步驟中發佈偵測器模型時使用的名稱。

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

這會建立具有類似以下內容之內容的檔案 (motorDetectorModel.json)。

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

傳送輸入以在 中測試偵測器模型 AWS IoT Events

有幾種方式可在 中接收遙測資料 AWS IoT Events （請參閱 在 中接收資料和觸發動作的支援動作 AWS IoT Events)。本主題說明如何在 AWS IoT 主控台中建立 AWS IoT 規則，將訊息做為輸入轉送至 AWS IoT Events 偵測器。您可以使用 AWS IoT 主控台的 MQTT 用戶端來傳送測試訊息。當您的裝置可以使用訊息 AWS IoT 代理程式傳送 MQTT 訊息 AWS IoT Events 時，您可以使用此方法將遙測資料傳送到 。

傳送輸入以測試偵測器模型

1. 開啟 AWS IoT Core 主控台。在左側導覽窗格的管理下，選擇訊息路由，然後選擇規則。
2. 選擇右上角的建立規則。
3. 在建立規則頁面上，完成下列步驟：

   1. 步驟 1. 指定規則屬性。完成下列欄位：
      - 規則名稱。輸入規則的名稱，例如 `MyIoTEventsRule`。

        > **Note**
        > 請勿使用空格。

      - 規則描述。這是選用的。
      - 選擇下一步。
   2. 步驟 2. 設定 SQL 陳述式。完成下列欄位：
      - SQL 版本。從清單中選擇適當的選項。
      - SQL 陳述式。輸入 **`SELECT *, topic(2) as motorid FROM 'motors/+/status'`**。

      選擇下一步。
   3. 步驟 3. 連接規則動作。在規則動作區段中，完成下列操作：
      - 動作 1。選取 IoT 事件。出現下列欄位：

        a. 輸入名稱。從清單中選擇適當的選項。如果您的輸入未顯示，請選擇重新整理。

        若要建立新的輸入，請選擇建立 IoT 事件輸入。完成下列欄位：

        - 輸入名稱。輸入 `PressureInput`。
        - 描述。這是選用的。
        - 上傳 JSON 檔案。上傳 JSON 檔案的副本。如果您沒有 檔案，此畫面上會有範例檔案的連結。程式碼包括：

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

```
}
```

- 選擇輸入屬性。選取適當的 （選項）。
- Tags (標籤)。這是選用的。

選擇建立。

返回建立規則畫面並重新整理輸入名稱欄位。選取您剛建立的輸入。

b. 批次模式。這是選用的。如果承載是一系列訊息，請選取此選項。

c. 訊息 ID。此為選用操作，但建議您採用。

d. IAM 角色。從清單中選擇適當的角色。如果未列出角色，請選擇建立新角色。

輸入角色名稱，然後選擇建立。

若要新增另一個規則，請選擇新增規則動作

- 錯誤動作。此區段為選用。若要新增動作，請選擇新增錯誤動作，然後從清單中選擇適當的動作。

完成出現的欄位。

- 選擇下一步。

4. 步驟 4. 檢閱和建立。檢閱畫面上的資訊，然後選擇建立。

4. 在左側導覽窗格的測試下，選擇 MQTT 測試用戶端。

5. 請選擇 Publish to a topic (發佈至主題)。完成下列欄位：

- 主題名稱。輸入名稱以識別訊息，例如 `motors/Fulton-A32/status`。
- 訊息承載。輸入下列資料：

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

> **Note**
>
> 每次發佈新訊息`messageId`時變更 。

6. 對於發佈，請保持主題相同，但將承載"pressure"中的 變更為大於您在偵測器模型中指定的閾值的值 （ 例如 **85**)。
7. 選擇發布。

您建立的偵測器執行個體會產生並傳送 Amazon SNS 訊息給您。繼續傳送壓力讀數高於或低於壓力閾值 （ 此範例為 70) 的訊息，以查看運作中的偵測器。

在此範例中，您必須傳送三個壓力讀數低於閾值的訊息，以轉換回正常狀態，並收到指出過壓條件已清除的 Amazon SNS 訊息。回到正常狀態後，壓力讀數超過限制的訊息會導致偵測器進入危險狀態，並傳送指出該條件的 Amazon SNS 訊息。

現在您已建立簡單的輸入和偵測器模型，請嘗試下列操作。

- 在 主控台上查看更多偵測器模型範例 （ 範本 ）。
- 遵循中的步驟使用 CLI 建立兩個狀態的 AWS IoT Events 偵測器，使用 建立輸入和偵測器模型 AWS CLI
- 了解 事件篩選、轉換和處理事件資料的表達式中使用的 詳細資訊。
- 了解 在 中接收資料和觸發動作的支援動作 AWS IoT Events。
- 如果某些項目無法運作，請參閱 故障診斷 AWS IoT Events。

的最佳實務 AWS IoT Events

遵循這些最佳實務以取得最大效益 AWS IoT Events。

主題

在開發 AWS IoT Events 偵測器模型時啟用 Amazon CloudWatch 記錄

Amazon CloudWatch AWS 會即時監控您的 AWS 資源和您在 上執行的應用程式。透過 CloudWatch，您可以全面了解資源使用、應用程式效能和運作狀態。當您開發或偵錯 AWS IoT Events 偵測器模型時，CloudWatch 可協助您了解 AWS IoT Events 正在做什麼，以及它遇到的任何錯誤。

啟用 CloudWatch

1. 如果您尚未設定 的許可 AWS IoT Events建立，請依照 中的步驟建立具有連接政策的角色，該政策會授予建立和管理 CloudWatch 日誌的許可 AWS IoT Events。
2. 前往 AWS IoT Events 主控台。
3. 在導覽窗格中，選擇設定。
4. 在設定頁面上，選擇編輯。
5. 在編輯記錄選項頁面的記錄選項區段中，執行下列動作：
   a. 針對詳細程度，選取選項。
   b. 針對選取角色，選取具有足夠許可的角色，以執行您選擇的記錄動作。
   c. （選用）如果您為詳細程度選擇偵錯，您可以執行下列動作來新增偵錯目標：
      i. 在偵錯目標下，選擇新增模型選項。
      ii. 輸入偵測器模型名稱和（選用）KeyValue，以指定要記錄的偵測器模型和特定偵測器（執行個體）。
6. 選擇更新。

您的記錄選項已成功更新。

定期發佈以在 AWS IoT Events 主控台中工作時儲存偵測器模型

當您使用 AWS IoT Events 主控台時，您的進行中工作會儲存在瀏覽器本機。不過，您必須選擇發佈以儲存偵測器模型 AWS IoT Events。發佈偵測器模型後，您發佈的工作將可在您用來存取帳戶的任何瀏覽器中使用。

> **Note**
>
> 如果您不發佈工作，則不會儲存工作。發佈偵測器模型後，您無法變更其名稱。不過，您可以繼續修改其定義。

AWS IoT Events 使用案例的教學課程

AWS IoT Events 教學課程提供涵蓋各種層面的程序集合 AWS IoT Events，從基本設定到更具體的使用案例。每個教學課程都顯示實際案例的範例，協助您建立偵測器模型、設定輸入、設定動作，以及與其他 AWS 服務整合以建立強大 IoT 解決方案的真實世界技能。

本章說明如何：

- 取得協助以決定偵測器模型中要包含哪些狀態，並判斷您是否需要一個或數個偵測器執行個體。
- 遵循使用 的範例 AWS CLI。
- 建立輸入以從裝置和偵測器模型接收遙測資料，以監控和報告傳送該資料的裝置狀態。
- 檢閱輸入、偵測器模型和服務的限制。 AWS IoT Events
- 請參閱更複雜的偵測器模型範例，其中包含註解。

主題

使用 AWS IoT Events 監控您的 IoT 裝置

您可以使用 AWS IoT Events 來監控裝置或程序，並根據重大事件採取行動。若要這麼做，請依照下列基本步驟進行：

建立輸入

您必須有一種方式，您的裝置和程序才能取得遙測資料 AWS IoT Events。您可以透過將訊息做為輸入傳送到 來執行此操作 AWS IoT Events。您可以透過多種方式將訊息做為輸入來傳送：

- 使用 BatchPutMessage 操作。
- 定義`iotEvents`規則引擎的規則動作。 AWS IoT Core規則動作會將您輸入的訊息資料轉送至 AWS IoT Events。
- 在 中 AWS IoT Analytics，使用 CreateDataset 操作建立具有 的資料集`contentDeliveryRules`。這些規則會指定自動傳送資料集內容的 AWS IoT Events 輸入。

- 在 AWS IoT Events 偵測器模型的 `onInput`、 `onExit`或 `transitionEvents`事件中定義 iotEvents 動作。偵測器模型執行個體和啟動動作之事件的相關資訊，會以您指定的名稱做為輸入傳回系統。

在您的裝置以這種方式開始傳送資料之前，您必須定義一或多個輸入。若要這樣做，請為每個輸入命名，並指定輸入監視器在傳入訊息資料中的哪些欄位。 會從許多來源 AWS IoT Events 以 JSON 承載的形式接收其輸入。每個輸入都可以單獨使用，也可以與其他輸入結合，以偵測更複雜的事件。

建立偵測器模型

使用 狀態定義偵測器模型 （設備或程序的模型）。針對每個狀態，您可以定義評估傳入輸入的條件 (布林值) 邏輯，以偵測重大事件。偵測到事件時，可以變更狀態，或使用其他 AWS 服務啟動自訂建置或預先定義的動作。您可以定義其他事件，在進入或退出狀態時啟動動作，也可以選擇在符合條件時啟動動作。

在本教學課程中，您會在模型進入或退出特定狀態時傳送 Amazon SNS 訊息做為動作。

監控裝置或程序

如果您正在監控多個裝置或程序，請在每個輸入中指定欄位，以識別輸入來源的特定裝置或程序。（請參閱 中的 `key` 欄位`CreateDetectorModel`。）識別新裝置時 （在由 識別的輸入欄位中看到新值`key`），會建立偵測器。（每個偵測器都是偵測器模型的執行個體。）然後，新的偵測器會繼續回應來自該裝置的輸入，直到其偵測器模型更新或刪除為止。

如果您正在監控單一程序 （即使有多個裝置或子程序正在傳送輸入），則不會指定唯一的識別`key`欄位。在此情況下，會在第一個輸入到達時建立單一偵測器 （執行個體）。

將訊息做為輸入傳送到偵測器模型

有幾種方式可以從裝置或程序傳送訊息，做為 AWS IoT Events 偵測器的輸入，而不需要您對訊息執行其他格式設定。在本教學課程中，您會使用 AWS IoT 主控台為轉送訊息資料的 AWS IoT Core 規則引擎撰寫AWS IoT Events 動作規則 AWS IoT Events。若要這樣做，您可以依名稱識別輸入。然後，您繼續使用 AWS IoT 主控台來產生一些做為輸入轉送的訊息 AWS IoT Events。

如何知道偵測器模型中需要哪些狀態？

若要判斷偵測器模型應有的狀態，請先決定您可以採取的動作。例如，如果您的汽車在油品上執行，您會在開始行程時查看油量計，以查看是否需要重新注油。您在這裡有一個動作：告知驅動程式「取得瓦斯」。您的偵測器模型需要兩種狀態：「汽車不需要燃料」和「汽車需要燃料」。一般而言，您想要

為每個可能的動作定義一個狀態，以及在不需要動作時再定義一個狀態。即使動作本身更複雜，這也有效。例如，您可能想要查詢並包含尋找最接近的瓦斯站或最便宜價格的資訊，但當您傳送訊息到「取得瓦斯」時，就會這樣做。

若要決定接下來要進入的狀態，您可以查看輸入。輸入包含決定您應處於何種狀態所需的資訊。若要建立輸入，請在裝置或程序傳送的訊息中選取一或多個欄位，以協助您做出決定。在此範例中，您需要一個輸入，告訴您目前的油位 (「百分比滿」)。也許您的汽車正在傳送數個不同的訊息給您，每個訊息都有數個不同的欄位。若要建立此輸入，您必須選取報告目前瓦斯表關卡的訊息和欄位。您即將進行的旅程長度 (「距離目的地」) 可以進行硬式編碼，以保持簡單；您可以使用平均旅程長度。您將根據輸入進行一些計算 （該百分比完全轉譯多少 gallons？ 是大於您可以行駛之英里的平均行程長度，取決於您擁有的 gallons 和您的平均 "miles per gallon")。您會執行這些計算並在事件中傳送訊息。

到目前為止，您有兩個狀態和一個輸入。您需要處於第一個狀態的事件，該事件會根據輸入執行計算，並決定是否要進入第二個狀態。這是一個轉換事件。 (`transitionEvents` 位於狀態`onInput`的事件清單中。 收到處於此第一個狀態的輸入時，如果`condition`符合事件的 ，事件會執行轉換為第二個狀態。) 當您達到第二個狀態時，您會在進入 狀態後立即傳送訊息。 (您使用 `onEnter`事件。 進入第二個狀態時，此事件會傳送訊息。 不需要等待另一個輸入到達。) 還有其他類型的事件，但這只是簡單範例所需的。

其他類型的事件是 `onExit`和 `onInput`。一旦收到輸入並符合條件，`onInput`事件就會執行指定的動作。當操作結束其目前狀態且符合條件時，`onExit`事件會執行指定的動作。

您有任何遺漏嗎？ 是，如何回到第一個「汽車不需要燃料」狀態？ 填滿您的坦克之後，輸入會顯示已滿的坦克。在第二個狀態下，您需要轉換事件回到接收輸入時 （在第二個狀態`onInput:`的事件中） 發生的第一個狀態。如果其計算結果顯示您現在有足夠的瓦斯可以讓您前往目的地，則它應該會轉返到第一個狀態。

這就是基本概念。有些偵測器模型透過新增可反映重要輸入的狀態來變得更複雜，而不只是可能的動作。例如，您可能在偵測器模型中有三種狀態，可追蹤溫度：「正常」狀態、「過熱」狀態，以及「潛在問題」狀態。當溫度超過特定層級，但尚未變得過熱時，您會轉換為潛在問題狀態。除非警示保持在此溫度超過 15 分鐘，否則您不想傳送警示。如果在此之前溫度恢復正常，偵測器會轉換回正常狀態。如果計時器過期，偵測器會轉換為過熱狀態並傳送警示，只是要小心。您可以使用變數和一組更複雜的事件條件來執行相同的動作。但通常使用另一個狀態來儲存計算結果會比較容易。

如何得知您需要一個或數個偵測器的執行個體？

若要決定您需要多少執行個體，請自問「您有興趣知道什麼？」 假設您想要知道今天天氣如何。是否下雨 （狀態）？ 您需要使用雨傘 （動作） 嗎？ 您可以有一個報告溫度的感應器、另一個報告濕度的感應器，以及其他報告大氣壓力、風速和方向以及降水的感應器。但是，您必須同時監控所有這些感應

器，以判斷天氣狀態（雨水、雪水、過度投射、陽光）和要採取的適當動作（拿雨傘或擦太陽霜）。儘管感應器數量有限，您希望一個偵測器執行個體監控天氣狀態，並通知您要採取的動作。

但是，如果您是區域的天氣預測器，則可能會有多個此類感應器陣列的執行個體，位於整個區域的不同位置。每個位置的人員都需要知道該位置的天氣狀況。在這種情況下，您需要偵測器的多個執行個體。每個位置中每個感應器報告的資料必須包含您指定為 欄位key的欄位。此欄位可讓 AWS IoT Events 建立區域的偵測器執行個體，然後在偵測器執行個體持續到達時，繼續將此資訊路由至該偵測器執行個體。不再有受損的頭髮或太陽鼻涕！

基本上，如果您有一個情況（一個程序或一個位置）需要監控，則需要一個偵測器執行個體。如果您有許多需要監控的情況（位置、程序），則需要多個偵測器執行個體。

使用 CLI 建立兩個狀態的 AWS IoT Events 偵測器

在此範例中，我們使用 AWS CLI 命令呼叫 AWS IoT Events APIs，以建立模擬引擎兩個狀態的偵測器：正常狀態和過壓條件。

當引擎中測量到的壓力超過特定閾值時，模型會轉換為過壓狀態，並傳送 Amazon Simple Notification Service (Amazon SNS) 訊息來提醒技術人員注意 條件。當壓力低於連續三個壓力讀數的閾值時，模型會返回正常狀態，並傳送另一個 Amazon SNS 訊息，以確認條件已清除。我們需要連續三個低於壓力閾值的讀數，以消除在非線性復原階段或一次性異常復原讀取的情況下可能發生的過壓/正常訊息停滯。

以下是建立偵測器的步驟概觀。

建立輸入。

若要監控您的裝置和程序，它們必須能夠將遙測資料匯入 AWS IoT Events。這可透過將訊息作為輸入傳送到 來完成 AWS IoT Events。您可以數種方式來執行此動作：

- 使用 BatchPutMessage 操作。此方法很簡單，但需要您的裝置或程序能夠透過 SDK 或 存取 AWS IoT Events API AWS CLI。
- 在 中 AWS IoT Core，為轉送訊息資料的 AWS IoT Core 規則引擎撰寫AWS IoT Events 動作規則 AWS IoT Events。這會依名稱識別輸入。如果您的裝置或程序可以或已經透過 傳送訊息，請使用此方法 AWS IoT Core。此方法通常需要較少的裝置運算能力。
- 在 中 AWS IoT Analytics，使用 CreateDataset 操作建立具有 的資料集`contentDeliveryRules`，以指定 AWS IoT Events 輸入，其中會自動傳送資料集內容。如果您想要根據彙總或分析的資料來控制裝置或程序，請使用此方法 AWS IoT Analytics。

您必須先定義一或多個輸入，您的裝置才能以這種方式傳送資料。若要這樣做，請為每個輸入命名，並指定輸入監控的傳入訊息資料中的哪些欄位。

建立偵測器模型

使用 狀態建立偵測器模型 （設備或程序的模型）。針對每個狀態，定義評估傳入輸入的條件式（布林值）邏輯，以偵測重大事件。偵測到事件時，可以變更狀態，或使用其他 AWS 服務啟動自訂建置或預先定義的動作。您可以定義其他事件，在進入或退出狀態時啟動動作，也可以選擇在符合條件時啟動動作。

監控數個裝置或程序

如果您正在監控多個裝置或程序，並且想要分別追蹤每個裝置或程序，請在每個輸入中指定 欄位，以識別輸入來源的特定裝置或程序。請參閱 中的 `key` 欄位`CreateDetectorModel`。識別新裝置時（在由 識別的輸入欄位中看到新值`key`），會建立偵測器執行個體。新的偵測器執行個體會繼續回應來自該特定裝置的輸入，直到其偵測器模型更新或刪除為止。您擁有與輸入`key`欄位中有唯一值一樣多的唯一偵測器（執行個體）。

監控單一裝置或程序

如果您正在監控單一程序（即使有多個裝置或子程序正在傳送輸入），則不會指定唯一的識別`key`欄位。在此情況下，會在第一個輸入到達時建立單一偵測器（執行個體）。例如，您可能在房子的每個房間都有溫度感測器，但只有一個 HVAC 單元來加熱或冷卻整個房子。因此，即使每個房間佔用者都希望投票（輸入）優先，您也只能以單一程序控制。

從裝置或程序傳送訊息，做為偵測器模型的輸入

我們描述了從裝置或程序傳送訊息的數種方法，做為輸入中 AWS IoT Events 偵測器的輸入。在您建立輸入並建置偵測器模型之後，您就可以開始傳送資料。

> **Note**
>
> 當您建立偵測器模型或更新現有模型時，新偵測器模型或更新的偵測器模型開始接收訊息和建立偵測器（執行個體）需要幾分鐘的時間。如果偵測器模型已更新，在此期間，您可能會繼續看到根據先前版本的行為。

主題

- 建立 AWS IoT Events 輸入以擷取裝置資料
- 建立偵測器模型以代表 中的裝置狀態 AWS IoT Events
- 在 中將訊息做為輸入傳送到偵測器 AWS IoT Events

建立 AWS IoT Events 輸入以擷取裝置資料

設定 的輸入時 AWS IoT Events，您可以利用 AWS CLI 來定義裝置如何通訊感應器資料。例如，如果您的裝置傳送具有馬達識別符和感應器讀數的 JSON 格式訊息，您可以透過建立從訊息映射特定屬性的輸入來擷取此資料，例如壓力和馬達 ID。程序從定義 JSON 檔案中的輸入、指定相關資料點，以及使用 AWS CLI 註冊輸入開始 AWS IoT Events。這可讓 根據即時感應器資料 AWS IoT 監控和回應關鍵條件。

例如，假設您的裝置以下列格式傳送訊息。

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

您可以使用下列 AWS CLI 命令建立輸入來擷取`pressure`資料和 `motorid`（識別傳送訊息的特定裝置）。

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

檔案`pressureInput.json`包含下列項目。

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

當您建立自己的輸入時，請記得先從裝置或程序收集範例訊息做為 JSON 檔案。您可以使用它們從主控台或 CLI 建立輸入。

建立偵測器模型以代表 中的裝置狀態 AWS IoT Events

在 中建立 AWS IoT Events 輸入以擷取裝置資料，您input根據報告來自馬達壓力資料的訊息建立 。若要繼續範例，以下是偵測器模型，可回應馬達中的過壓事件。

您建立兩種狀態：「Normal」和「Dangerous」。每個偵測器（執行個體）都會在建立時進入「Normal」狀態。當具有key「motorid」唯一值的輸入到達時，就會建立執行個體。

如果偵測器執行個體收到 70 或更高的壓力讀數，它會進入「Dangerous」狀態，並傳送 Amazon SNS 訊息作為警告。如果連續三個輸入的壓力讀數回到正常（小於 70)，偵測器會回到「Normal」狀態，並傳送另一個 Amazon SNS 訊息作為全部清除。

此範例偵測器模型假設您已建立兩個 Amazon SNS 主題，其 Amazon Resource Name (ARNs) 在定義中顯示為 `"targetArn": "arn:aws:sns:us-east-1:123456789012:underPressureAction"`和 `"targetArn": "arn:aws:sns:us-east-1:123456789012:pressureClearedAction"`。

如需詳細資訊，請參閱 Amazon Simple Notification Service 開發人員指南，以及 Amazon Simple Notification Service API 參考中 CreateTopic 操作的文件。

此範例也假設您已建立具有適當許可的 AWS Identity and Access Management (IAM) 角色。此角色的 ARN 在偵測器模型定義中顯示為 `"roleArn": "arn:aws:iam::123456789012:role/IoTEventsRole"`。請依照 中的步驟設定 的許可 AWS IoT Events建立此角色，並在偵測器模型定義的適當位置複製角色的 ARN。

您可以使用下列 AWS CLI 命令建立偵測器模型。

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

檔案`"motorDetectorModel.json"`包含下列項目。

```
{
  "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"
}
```

在 中將訊息做為輸入傳送到偵測器 AWS IoT Events

您現在已定義輸入，識別從裝置傳送的訊息中的重要欄位（請參閱 建立 AWS IoT Events 輸入以擷取裝置資料)。在上一節中，您建立了 detector model來回應馬達中的過壓事件（請參閱 建立偵測器模型以代表 中的裝置狀態 AWS IoT Events)。

若要完成範例，請從裝置（在此情況下為 AWS CLI 已安裝 的電腦）傳送訊息，做為偵測器的輸入。

> **Note**
>
> 當您建立偵測器模型或更新現有的偵測器模型時，需要幾分鐘的時間，新的或更新的偵測器模型才會開始接收訊息並建立偵測器（執行個體）。如果您更新偵測器模型，在此期間，您可能會繼續看到根據先前版本的行為。

使用下列 AWS CLI 命令來傳送訊息，其中包含超出閾值的資料。

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

檔案「highPressureMessage.json」包含下列項目。

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

您必須在每個傳送的訊息messageId中變更 。如果您不變更訊息， AWS IoT Events 系統會刪除重複訊息。如果訊息與過去五分鐘內傳送messageID的其他訊息相同，則 AWS IoT Events 忽略訊息。

此時，會建立偵測器（執行個體）來監控馬達 的事件"Fulton-A32"。此偵測器會在建立時進入 "Normal" 狀態。但是，因為我們傳送了超過閾值的壓力值，它會立即轉換為 "Dangerous" 狀態。這樣做時，偵測器會傳送訊息至 ARN 為 的 Amazon SNS 端點arn:aws:sns:us-east-1:123456789012:underPressureAction。

執行下列 AWS CLI 命令來傳送訊息，其中包含低於壓力閾值的資料。

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

檔案normalPressureMessage.json包含下列項目。

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

每次在五分鐘內叫用 BatchPutMessage命令時，都必須變更 檔案messageId中的 。再傳送訊息兩次。訊息傳送三次後，馬達「Fulton-A32」的偵測器（執行個體）會傳送訊息至 Amazon SNS 端點"arn:aws:sns:us-east-1:123456789012:pressureClearedAction"，然後重新進入 "Normal" 狀態。

> **Note**
>
> 您可以使用 一次傳送多則訊息BatchPutMessage。不過，無法保證處理這些訊息的順序。若要保證訊息（輸入）會依序處理，請一次傳送一個訊息，並在每次呼叫 API 時等待成功回應。

以下是本節中所述偵測器模型範例所建立的 SNS 訊息承載範例。

事件發生時「違反壓力閾值」

```
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"
}
```

事件發生時「正常壓力已還原」

```
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"
}
```

如果您已定義任何計時器，SNS 訊息承載中也會顯示其目前狀態。

訊息承載包含訊息傳送時偵測器（執行個體）狀態的相關資訊（即 SNS 動作執行時）。您可以使用 https://docs.aws.amazon.com/iotevents/latest/apireference/API_iotevents-data_DescribeDetector.html操作來取得偵測器狀態的類似資訊。

AWS IoT Events 偵測器模型限制

建立偵測器模型時，需要考量下列事項。

如何使用 **`actions`** 欄位

`actions` 欄位是 物件的清單。您可以有多個物件，但每個物件只允許一個動作。

Example

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

如何使用 **condition** 欄位

和 condition在其他情況下transitionEvents為選用項目。

如果 condition 欄位不存在，則相當於 "condition": true。

條件表達式的評估結果應為布林值。如果結果不是布林值，則相當於 false，且 不會啟動 actions或 轉換為事件中nextState指定的 。

變數值的可用性

根據預設，如果變數的值是在事件中設定，則無法使用其新值，或用於評估相同群組中其他事件的條件。新值無法用於相同 onInput、 onEnter或 onExit欄位中的事件條件。

在偵測器模型定義中設定 evaluationMethod 參數，以變更此行為。將 evaluationMethod 設定為 時SERIAL，會更新變數，並依事件定義的順序評估事件條件。否則，當 evaluationMethod 設為 BATCH或預設為 時，狀態內的變數會更新，而且只有在評估所有事件條件後，才會執行狀態內的事件。

在 "Dangerous" 狀態中，在 onInput 欄位中，"Pressure Okay"當條件滿足時（當目前的輸入壓力小於或等於 70 時），"$variable.pressureThresholdBreached" 會以 1 遞減。

```
      {
        "eventName": "Pressure Okay",
        "condition": "$input.PressureInput.sensorData.pressure <= 70",
```

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

當 "$variable.pressureThresholdBreached"達到 0 時（亦即當偵測器收到三個小於或等於 70 的連續壓力讀數時），偵測器應轉返 "Normal" 狀態。中的"BackToNormal"事件transitionEvents必須測試"$variable.pressureThresholdBreached"小於或等於 1（非 0)，並再次驗證 提供的目前值"$input.PressureInput.sensorData.pressure"小於或等於 70。

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

否則，如果條件只測試變數的值，則兩個正常讀數後接過壓力讀數會滿足條件並轉換回 "Normal" 狀態。條件正在查看之前處理輸入時"$variable.pressureThresholdBreached"所提供的值。變數的值會在"Overpressurized"事件中重設為 3，但請記住，此新值尚不適用於任何condition。

根據預設，每次控制項進入 onInput 欄位時， condition只能看到變數在處理輸入開始時的值，然後再由 中指定的任何動作變更onInput。onEnter 和 也是如此onExit。當我們進入或退出 狀態時對變數所做的任何變更，都不適用於相同 onEnter或 onExit 欄位中指定的其他條件。

更新偵測器模型時的延遲

如果您更新、刪除和重新建立偵測器模型（請參閱 UpdateDetectorModel)，則在刪除所有產生的偵測器（執行個體）之前會有一些延遲，並使用新模型重新建立偵測器。它們會在新偵測器模型生效且新輸入到達後重新建立。在此期間，先前版本的偵測器模型可能會繼續處理輸入。在此期間，您可能會繼續收到先前偵測器模型定義的提醒。

輸入金鑰中的空間

輸入索引鍵中允許空格，但索引鍵的參考必須以反引號括住，包括輸入屬性的定義，以及表達式中參考索引鍵的值時。例如，指定訊息承載，如下所示：

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

使用下列項目來定義輸入。

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

在條件式表達式中，您還必須使用反引號參考任何此類索引鍵的值。

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

註解範例：使用 的 HVAC 溫度控制 AWS IoT Events

下列一些範例 JSON 檔案內嵌註解，這會使它們變成無效的 JSON。這些範例的完整版本，不含註解，可在 取得範例：搭配 使用 HVAC 溫度控制 AWS IoT Events。

此範例實作調溫器控制模型，讓您能夠執行下列動作。

- 僅定義一個偵測器模型，可用於監控和控制多個區域。系統會為每個區域建立偵測器執行個體。
- 從每個控制區域中的多個感應器擷取溫度資料。

- 變更區域的溫度設定點。
- 為每個區域設定操作參數，並在執行個體使用時重設這些參數。
- 從 區域動態新增或刪除感應器。
- 指定最小執行時間以保護加熱和冷卻單位。
- 拒絕異常感應器讀數。
- 如果任何一個感應器報告高於或低於指定閾值的溫度，請定義立即進行加熱或冷卻的緊急設定點。
- 報告異常讀數和溫度峰值。

主題

中的偵測器模型輸入定義 AWS IoT Events

我們希望建立一個偵測器模型，可用來監控和控制數個不同區域的溫度。每個區域都可以有數個報告溫度的感應器。我們假設每個區域都由一個加熱單元和一個冷卻單元提供，可以開啟或關閉以控制區域中的溫度。每個區域都由一個偵測器執行個體控制。

由於我們監控和控制的不同區域可能會有不同的特性，需要不同的控制參數，因此我們定義 `'seedTemperatureInput'`來為每個區域提供這些參數。當我們傳送其中一個輸入訊息到 時 AWS IoT Events，會建立新的偵測器模型執行個體，其中包含我們在該區域中要使用的參數。以下是該輸入的定義。

CLI 命令：

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

檔案： 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" }
    ]
  }
}
```

回應：

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

備註

- 系統會針對任何訊息中'areaId'收到的每個唯一 建立新的偵測器執行個體。請參閱 'areaDetectorModel'定義中的 'key' 欄位。
- 在為區域啟用加熱或冷卻裝置'desiredTemperature''allowedError'之前， 的平均溫度可能會與 不同。
- 如果有任何感應器報告高於 的溫度'rangeHigh'，偵測器會報告峰值並立即啟動冷卻裝置。

- 如果有任何感應器報告低於 的溫度'rangeLow'，偵測器會報告峰值並立即啟動加熱裝置。
- 如果有任何感應器報告高於 'anomalousHigh' 或低於 的溫度'anomalousLow'，偵測器會報告異常感應器讀數，但會忽略報告的溫度讀數。
- 'sensorCount' 告訴偵測器該區域報告了多少個感應器。偵測器透過為每個接收到的溫度提供適當的權重因數來計算區域中的平均溫度。因此，偵測器不需要追蹤每個感應器報告的內容，並視需要動態變更感應器的數量。不過，如果個別感應器離線，偵測器就不會知道這一點或為其提供限額。我們建議您建立另一個偵測器模型，專門用於監控每個感應器的連線狀態。有兩個互補偵測器模型可簡化兩者的設計。
- 'noDelay' 值可以是 true或 false。開啟加熱或冷卻裝置後，應該保持開啟狀態一段最短時間，以保護裝置的完整性並延長其操作壽命。如果 'noDelay' 設定為 false，偵測器執行個體會在關閉冷卻和加熱單元之前強制執行延遲，以確保它們在最短時間內執行。由於我們無法使用變數值來設定計時器，因此偵測器模型定義中已硬式編碼延遲的秒數。

'temperatureInput' 用於將感應器資料傳輸到偵測器執行個體。

CLI 命令：

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

檔案： temperatureInput.json

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

回應：

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

備註

- 範例偵測器執行個體'sensorId'不會使用 來直接控制或監控感應器。它會自動傳遞到偵測器執行個體傳送的通知。從那裡，它可用於識別失敗的感應器（例如，定期傳送異常讀數的感應器可能即將失敗）或離線的感應器（當它用作監控裝置活動訊號的其他偵測器模型的輸入時）。如果區域讀數定期與平均值不同，'sensorId'也有助於識別區域中的暖區域或冷區域。
- 'areaId' 用於將感應器的資料路由到適當的偵測器執行個體。系統會針對任何訊息中'areaId'接收的每個唯一 建立偵測器執行個體。請參閱 'areaDetectorModel'定義中的 'key' 欄位。

建立 AWS IoT Events 偵測器模型定義

此'areaDetectorModel'範例具有內嵌註解。

CLI 命令：

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

檔案： areaDetectorModel.json

```
{
  "detectorModelName": "areaDetectorModel",
  "detectorModelDefinition": {
    "states": [
      {
        "stateName": "start",
        // In the 'start' state we set up the operation parameters of the new detector
 instance.
        //   We get here when the first input message arrives. If that is a
 'seedTemperatureInput'
        //   message, we save the operation parameters, then transition to the 'idle'
 state. If
        //   the first message is a 'temperatureInput', we wait here until we get a
```

```
        //   'seedTemperatureInput' input to ensure our operation parameters are set.
We can
        //   also reenter this state using the 'BatchUpdateDetector' API. This enables
us to
        //   reset the operation parameters without needing to delete the detector
instance.
        "onEnter": {
          "events": [
            {
              "eventName": "prepare",
              "condition": "true",
              "actions": [
                {
                  "setVariable": {
                    // initialize 'sensorId' to an invalid value (0) until an actual
sensor reading
                    //   arrives
                    "variableName": "sensorId",
                    "value": "0"
                  }
                },
                {
                  "setVariable": {
                    // initialize 'reportedTemperature' to an invalid value (0.1) until
an actual
                    //   sensor reading arrives
                    "variableName": "reportedTemperature",
                    "value": "0.1"
                  }
                },
                {
                  "setVariable": {
                    // When using 'BatchUpdateDetector' to re-enter this state, this
variable should
                    //   be set to true.
                    "variableName": "resetMe",
                    "value": "false"
                  }
                }
              ]
            }
          ]
        },
        "onInput": {
```

```
          "transitionEvents": [
            {
              "eventName": "initialize",
              "condition": "$input.seedTemperatureInput.sensorCount > 0",
              // When a 'seedTemperatureInput' message with a valid 'sensorCount' is
received,
              //   we use it to set the operational parameters for the area to be
monitored.
              "actions": [
                {
                  "setVariable": {
                    "variableName": "rangeHigh",
                    "value": "$input.seedTemperatureInput.rangeHigh"
                  }
                },
                {
                  "setVariable": {
                    "variableName": "rangeLow",
                    "value": "$input.seedTemperatureInput.rangeLow"
                  }
                },
                {
                  "setVariable": {
                    "variableName": "desiredTemperature",
                    "value": "$input.seedTemperatureInput.desiredTemperature"
                  }
                },
                {
                  "setVariable": {
                    // Assume we're at the desired temperature when we start.
                    "variableName": "averageTemperature",
                    "value": "$input.seedTemperatureInput.desiredTemperature"
                  }
                },
                {
                  "setVariable": {
                    "variableName": "allowedError",
                    "value": "$input.seedTemperatureInput.allowedError"
                  }
                },
                {
                  "setVariable": {
                    "variableName": "anomalousHigh",
                    "value": "$input.seedTemperatureInput.anomalousHigh"
```

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

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


    {
      "stateName": "idle",
```

```
        "onInput": {
          "events": [
            {
              "eventName": "whatWasInput",
              "condition": "true",
              // By storing the 'sensorId' and the 'temperature' in variables, we make
them
              //   available in any messages we send out to report anomalies, spikes,
or just
              //   if needed for debugging.
              "actions": [
                {
                  "setVariable": {
                    "variableName": "sensorId",
                    "value": "$input.temperatureInput.sensorId"
                  }
                },
                {
                  "setVariable": {
                    "variableName": "reportedTemperature",
                    "value": "$input.temperatureInput.sensorData.temperature"
                  }
                }
              ]
            },
            {
              "eventName": "changeDesired",
              "condition": "$input.seedTemperatureInput.desiredTemperature !=
$variable.desiredTemperature",
              // This event enables us to change the desired temperature at any time by
sending a
              //   'seedTemperatureInput' message. But note that other operational
parameters are not
              //   read or changed.
              "actions": [
                {
                  "setVariable": {
                    "variableName": "desiredTemperature",
                    "value": "$input.seedTemperatureInput.desiredTemperature"
                  }
                }
              ]
            },
            {
```

```
            "eventName": "calculateAverage",
            "condition": "$input.temperatureInput.sensorData.temperature <
$variable.anomalousHigh && $input.temperatureInput.sensorData.temperature >
$variable.anomalousLow",
            // If a valid temperature reading arrives, we use it to update the
average temperature.
            //   For simplicity, we assume our sensors will be sending updates at
about the same rate,
            //   so we can calculate an approximate average by giving equal weight to
each reading we receive.
            "actions": [
              {
                "setVariable": {
                  "variableName": "averageTemperature",
                  "value": "((($variable.averageTemperature * ($variable.sensorCount
- 1)) + $input.temperatureInput.sensorData.temperature) / $variable.sensorCount)"
                }
              }
            ]
          }
        ],
        "transitionEvents": [
          {
            "eventName": "anomalousInputArrived",
            "condition": "$input.temperatureInput.sensorData.temperature >=
$variable.anomalousHigh || $input.temperatureInput.sensorData.temperature <=
$variable.anomalousLow",
            // When an anomalous reading arrives, send an MQTT message, but stay in
the current state.
            "actions": [
              {
                "iotTopicPublish": {
                  "mqttTopic": "temperatureSensor/anomaly"
                }
              }
            ],
            "nextState": "idle"
          },

          {
            "eventName": "highTemperatureSpike",
            "condition": "$input.temperatureInput.sensorData.temperature >
$variable.rangeHigh",
```

```
              // When even a single temperature reading arrives that is above the
'rangeHigh', take
              //   emergency action to begin cooling, and report a high temperature
spike.
              "actions": [
                {
                  "iotTopicPublish": {
                    "mqttTopic": "temperatureSensor/spike"
                  }
                },
                {
                  "sns": {
                    "targetArn": "arn:aws:sns:us-west-2:123456789012:coolOn"
                  }
                },
                {
                  "iotTopicPublish": {
                    "mqttTopic": "hvac/Cooling/On"
                  }
                },
                {
                  "setVariable": {
                    // This is necessary because we want to set a timer to delay the
shutoff
                    //   of a cooling/heating unit, but we only want to set the timer
when we
                    //   enter that new state initially.
                    "variableName": "enteringNewState",
                    "value": "true"
                  }
                }
              ],
              "nextState": "cooling"
            },

            {
              "eventName": "lowTemperatureSpike",
              "condition": "$input.temperatureInput.sensorData.temperature <
$variable.rangeLow",
              // When even a single temperature reading arrives that is below the
'rangeLow', take
              //   emergency action to begin heating, and report a low-temperature
spike.
              "actions": [
```

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

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

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

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

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


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

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

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

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

```
        "transitionEvents": [
          // Note that some tests of temperature values (for example, the test for an
anomalous value)
          //   must be placed here in the 'transitionEvents' because they work
together with the tests
          //   in the other conditions to ensure that we implement the proper
"if..elseif..else" logic.
          //   But each transition event must have a destination state ('nextState'),
and even if that
          //   is actually the current state, the "onEnter" events for this state
will be executed again.
          //   This is the reason for the 'enteringNewState' variable and related.
          {
            "eventName": "anomalousInputArrived",
            "condition": "$input.temperatureInput.sensorData.temperature >=
$variable.anomalousHigh || $input.temperatureInput.sensorData.temperature <=
$variable.anomalousLow",
            "actions": [
              {
                "iotTopicPublish": {
                  "mqttTopic": "temperatureSensor/anomaly"
                }
              }
            ],
            "nextState": "cooling"
          },

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

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

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

            {
              "eventName": "desiredTemperature",
              "condition": "((((($variable.averageTemperature * ($variable.sensorCount
- 1)) + $input.temperatureInput.sensorData.temperature) / $variable.sensorCount) <=
($variable.desiredTemperature - $variable.allowedError)) && $variable.goodToGo ==
true",
```

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


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

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

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

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

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

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

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

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

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

```
        }
      }

    ],

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

回應：

```
{
    "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"
    }
}
```

使用 BatchUpdateDetector 更新 AWS IoT Events 偵測器模型

您可以使用 BatchUpdateDetector操作將偵測器執行個體置於已知狀態，包括計時器和變數值。在下列範例中， BatchUpdateDetector操作會重設溫度監控和控制下區域的操作參數。此操作可讓您執行此操作，而無需刪除、重新建立或更新偵測器模型。

CLI 命令：

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

檔案： areaDM.BUD.json

```
{
  "detectors": [
```

```
    {
      "messageId": "0001",
      "detectorModelName": "areaDetectorModel",
      "keyValue": "Area51",
      "state": {
        "stateName": "start",
        "variables": [
          {
            "name": "desiredTemperature",
            "value": "22"
          },
          {
            "name": "averageTemperature",
            "value": "22"
          },
          {
            "name": "allowedError",
            "value": "1.0"
          },
          {
            "name": "rangeHigh",
            "value": "30.0"
          },
          {
            "name": "rangeLow",
            "value": "15.0"
          },
          {
            "name": "anomalousHigh",
            "value": "60.0"
          },
          {
            "name": "anomalousLow",
            "value": "0.0"
          },
          {
            "name": "sensorCount",
            "value": "12"
          },
          {
            "name": "noDelay",
            "value": "true"
          },
          {
```

```
            "name": "goodToGo",
            "value": "true"
          },
          {
            "name": "sensorId",
            "value": "0"
          },
          {
            "name": "reportedTemperature",
            "value": "0.1"
          },
          {
            "name": "resetMe",
            // When 'resetMe' is true, our detector model knows that we have reentered
 the 'start' state
            //   to reset operational parameters, and will allow the next valid
 temperature sensor
            //   reading to cause the transition to the 'idle' state.
            "value": "true"
          }
        ],
        "timers": [
        ]
      }
    }
  ]
}
```

回應：

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

在 中使用 BatchPutMessage 進行輸入 AWS IoT Events

Example 1

使用 BatchPutMessage操作來傳送"seedTemperatureInput"訊息，在溫度控制和監控下設定指定區域的操作參數。收到具有新 AWS IoT Events 的任何訊息都會建立新的"areaId"偵測器執行個體。但是，在收到新區域"seedTemperatureInput"的訊息之前，新的偵測器執行個體不會將狀態變更為 "idle"，並開始監控溫度和控制加熱或冷卻單位。

CLI 命令：

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

檔案： 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}"
    }
  ]
}
```

回應：

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

Example

2

使用 BatchPutMessage操作來傳送訊息"temperatureInput"，以報告指定控制和監控區域中感應器的溫度感應器資料。

CLI 命令：

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

檔案： temperatureExample.json

```
{
```

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

回應：

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

Example 3

使用 BatchPutMessage操作來傳送訊息"seedTemperatureInput"，以變更指定區域的所需溫度值。

CLI 命令：

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

檔案： seedSetDesiredTemp.json

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

回應：

```
{
```

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

在 中擷取 MQTT 訊息 AWS IoT Events

如果您的感應器運算資源無法使用 "BatchPutMessage" API，但可以使用輕量型 MQTT 用戶端將其資料傳送至 AWS IoT Core 訊息中介裝置，您可以建立 AWS IoT Core 主題規則，將訊息資料重新導向至 AWS IoT Events 輸入。以下是 AWS IoT Events 主題規則的定義，該規則會從 MQTT 主題取得 "areaId"和 "sensorId"輸入欄位，並從訊息承載"temp"欄位取得 "sensorData.temperature" 欄位，並將這些資料擷取到我們的 中 AWS IoT Events "temperatureInput"。

CLI 命令：

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

檔案： seedSetDesiredTemp.json

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

回應：【無】

如果感應器傳送有關 主題的訊息"update/temperature/Area51/03"，其中包含下列承載。

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

這會導致資料擷取至 AWS IoT Events ，如同已進行下列 "BatchPutMessage" API 呼叫一樣。

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

檔案： spoofExample.json

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

在 中產生 Amazon SNS 訊息 AWS IoT Events

以下是"Area51"偵測器執行個體產生的 SNS 訊息範例。

AWS IoT Events 可以與 Amazon SNS 整合，根據偵測到的事件產生和發佈通知。本節示範 AWS IoT Events 偵測器執行個體如何產生 Amazon SNS 訊息，特別是「Area51」偵測器。這些範例展示 AWS IoT Events 偵測器內各種狀態和事件觸發的 Amazon SNS 通知結構和內容，說明 AWS IoT Events 結合 Amazon SNS 進行即時提醒和通訊的能力。

```
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 中設定 DescribeDetector API AWS IoT Events

中的 `DescribeDetector` API AWS IoT Events 可讓您擷取特定偵測器執行個體的詳細資訊。此操作提供偵測器目前狀態、變數值和作用中計時器的洞見。透過使用此 API，您可以監控 AWS IoT Events 偵測器的即時狀態，促進 IoT 事件處理工作流程的偵錯、分析和管理。

CLI 命令：

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

回應：

```
{
    "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"
    }
}
```

使用 的 AWS IoT Core 規則引擎 AWS IoT Events

下列規則會將 AWS IoT Core MQTT 訊息重新發佈為影子更新請求訊息。我們假設為由偵測器模型控制的每個區域定義了供暖單位和冷卻單位的 AWS IoT Core 物件。在此範例中，我們定義了名為 "Area51HeatingUnit"和 的物件"Area51CoolingUnit"。

CLI 命令：

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

檔案： 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"
        }
```

```
      }
    ]
  }
}
```

回應：【空白】

CLI 命令：

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

檔案：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"
        }
      }
    ]
  }
}
```

回應：【空白】

CLI 命令：

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

檔案：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"
        }
      }
    ]
  }
}
```

回應：【空白】

CLI 命令：

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

檔案：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"
        }
      }
    ]
  }
}
```

回應：【空白】

在 中接收資料和觸發動作的支援動作 AWS IoT Events

AWS IoT Events 可以在偵測到指定的事件或轉換事件時觸發動作。您可以定義內建動作以使用計時器或設定變數，或將資料傳送至其他 AWS 資源。了解如何設定和自訂這些動作，以建立對各種 IoT 事件的自動回應。

> **Note**
>
> 當您在偵測器模型中定義動作時，您可以將表達式用於字串資料類型的參數。如需詳細資訊，請參閱表達式。

AWS IoT Events 支援下列動作，可讓您使用計時器或設定變數：

- `setTimer` 來建立計時器。
- `resetTimer` 以重設計時器。
- `clearTimer` 刪除計時器。
- `setVariable` 來建立變數。

AWS IoT Events 支援下列動作，可讓您使用 AWS 服務：

- `iotTopicPublish` 在 MQTT 主題上發佈訊息。
- `iotEvents` 將資料傳送至 AWS IoT Events 做為輸入值。
- `iotSiteWise` 將資料傳送至 AWS IoT SiteWise中的資產屬性。
- `dynamoDB` 將資料傳送至 Amazon DynamoDB 資料表。
- `dynamoDBv2` 將資料傳送至 Amazon DynamoDB 資料表。
- `firehose` 將資料傳送至 Amazon Data Firehose 串流。
- `lambda` 叫用 AWS Lambda 函數。
- `sns` 以推送通知的形式傳送資料。
- `sqs` 將資料傳送至 Amazon SQS 佇列。

使用 AWS IoT Events 內建計時器和變數動作

AWS IoT Events 支援下列動作，可讓您使用計時器或設定變數：

- `setTimer` 來建立計時器。
- `resetTimer` 以重設計時器。
- `clearTimer` 刪除計時器。
- `setVariable` 來建立變數。

設定計時器動作

Set timer action

`setTimer` 動作可讓您建立持續時間以秒為單位的計時器。

More information (2)

建立計時器時，您必須指定下列參數。

`timerName`

計時器的名稱。

`durationExpression`

（選用）計時器的持續時間，以秒為單位。

持續時間表達式的評估結果會四捨五入至最接近的整數。例如，如果您將計時器設定為 60.99 秒，持續時間表達式的評估結果為 60 秒。

如需詳細資訊，請參閱 AWS IoT Events API 參考中的 SetTimerAction。

重設計時器動作

Reset timer action

`resetTimer` 動作可讓您將計時器設定為先前評估的持續時間表達式結果。

More information (1)

重設計時器時，您必須指定下列參數。

`timerName`

計時器的名稱。

AWS IoT Events 當您重設計時器時， 不會重新評估持續時間表達式。

如需詳細資訊，請參閱 AWS IoT Events API 參考中的 ResetTimerAction。

清除計時器動作

Clear timer action

`clearTimer` 動作可讓您刪除現有的計時器。

More information (1)

刪除計時器時，您必須指定下列參數。

`timerName`

計時器的名稱。

如需詳細資訊，請參閱 AWS IoT Events API 參考中的 ClearTimerAction。

設定變數動作

Set variable action

`setVariable` 動作可讓您建立具有指定值的變數。

More information (2)

建立變數時，您必須指定下列參數。

`variableName`

變數的名稱。

`value`

變數的新值。

如需詳細資訊，請參閱 AWS IoT Events API 參考中的 SetVariableAction。

AWS IoT Events 使用其他 AWS 服務

AWS IoT Events 支援下列動作，可讓您使用 AWS 服務：

- `iotTopicPublish` 在 MQTT 主題上發佈訊息。
- `iotEvents` 將資料傳送至 AWS IoT Events 做為輸入值。
- `iotSiteWise` 將資料傳送至 AWS IoT SiteWise中的資產屬性。
- `dynamoDB` 將資料傳送至 Amazon DynamoDB 資料表。
- `dynamoDBv2` 將資料傳送至 Amazon DynamoDB 資料表。
- `firehose` 將資料傳送至 Amazon Data Firehose 串流。
- `lambda` 叫用 AWS Lambda 函數。
- `sns` 以推送通知的形式傳送資料。
- `sqs` 將資料傳送至 Amazon SQS 佇列。

> **Important**
>
> - 您必須為要使用的 AWS IoT Events 和 AWS 服務選擇相同的 AWS 區域。如需支援區域的清單，請參閱《Amazon Web Services 一般參考》中的 AWS IoT Events 端點和配額。
> - 當您為 AWS IoT Events 動作建立其他 AWS 資源時，必須使用相同的 AWS 區域。如果您切換 AWS 區域，存取 AWS 資源時可能會遇到問題。

根據預設， 會在 JSON 中為任何動作 AWS IoT Events 產生標準承載。此動作承載包含具有偵測器模型執行個體相關資訊的所有屬性值對，以及觸發動作的事件。若要設定動作承載，您可以使用內容表達式。如需詳細資訊，請參閱 篩選、轉換和處理事件資料的表達式和 AWS IoT Events API 參考中的承載資料類型。

AWS IoT Core

IoT topic publish action

AWS IoT Core 動作可讓您透過訊息代理程式發佈 MQTT AWS IoT 訊息。如需支援區域的清單，請參閱《Amazon Web Services 一般參考》中的 AWS IoT Core 端點和配額。

AWS IoT 訊息代理程式會透過將訊息從發佈 AWS IoT 用戶端傳送至訂閱用戶端來連接用戶端。如需詳細資訊，請參閱《 AWS IoT 開發人員指南》中的裝置通訊協定。

More information (2)

當您發佈 MQTT 訊息時，您必須指定下列參數。

`mqttTopic`

接收訊息的 MQTT 主題。

您可以使用偵測器模型中建立的變數或輸入值，在執行時間動態定義 MQTT 主題名稱。

`payload`

（選用）預設承載包含具有偵測器模型執行個體和事件觸發動作相關資訊的所有屬性值對。您也可以自訂承載。如需詳細資訊，請參閱 AWS IoT Events API 參考中的承載。

> **Note**
>
> 請確定連接至 AWS IoT Events 服務角色的政策授予 `iot:Publish`許可。如需詳細資訊，請參閱的身分和存取管理 AWS IoT Events。

如需詳細資訊，請參閱 AWS IoT Events API 參考中的 IotTopicPublishAction。

AWS IoT Events

IoT Events action

AWS IoT Events 動作可讓您將資料傳送至 AWS IoT Events 做為輸入。如需支援區域的清單，請參閱《Amazon Web Services 一般參考》中的 AWS IoT Events 端點和配額。

AWS IoT Events 可讓您監控設備或裝置機群操作失敗或變更，並在發生此類事件時觸發動作。如需詳細資訊，請參閱《 AWS IoT Events 開發人員指南》中的什麼是 AWS IoT Events？。

More information (2)

將資料傳送到 時 AWS IoT Events，您必須指定下列參數。

`inputName`

接收資料的 AWS IoT Events 輸入名稱。

`payload`

（選用）預設承載包含具有偵測器模型執行個體和事件觸發動作相關資訊的所有屬性值對。您也可以自訂承載。如需詳細資訊，請參閱 AWS IoT Events API 參考中的承載。

> **Note**
>
> 請確定連接至 AWS IoT Events 服務角色的政策授予 `iotevents:BatchPutMessage`許可。如需詳細資訊，請參閱的身分和存取管理 AWS IoT Events。

如需詳細資訊，請參閱 AWS IoT Events API 參考中的 IotEventsAction。

AWS IoT SiteWise

IoT SiteWise action

AWS IoT SiteWise 動作可讓您將資料傳送至 中的資產屬性 AWS IoT SiteWise。如需支援區域的清單，請參閱《Amazon Web Services 一般參考》中的 AWS IoT SiteWise 端點和配額。

AWS IoT SiteWise 是一項受管服務，可讓您大規模收集、組織和分析工業設備的資料。如需詳細資訊，請參閱《 AWS IoT SiteWise使用者指南》中的什麼是AWS IoT SiteWise ？。

More information (11)

當您將資料傳送至 中的資產屬性時 AWS IoT SiteWise，您必須指定下列參數。

> **Important**
>
> 若要接收資料，您必須使用 中的現有資產屬性 AWS IoT SiteWise。
>
> - 如果您使用 AWS IoT Events 主控台，則必須指定 `propertyAlias`來識別目標資產屬性。
> - 如果您使用 AWS CLI，則必須指定 `propertyAlias`或兩者`assetIdpropertyId`，並識別目標資產屬性。
>
> 如需詳細資訊，請參閱《AWS IoT SiteWise 使用者指南》中的將工業資料串流映射至資產屬性。

propertyAlias

（選用）資產屬性的別名。您也可以指定表達式。

assetId

（選用）具有指定屬性的資產 ID。您也可以指定表達式。

propertyId

（選用）資產屬性的 ID。您也可以指定表達式。

entryId

(選用) 此項目的唯一識別符。您可以使用項目 ID 來追蹤在失敗時，哪個資料項目會發生錯誤。預設值為新的唯一識別碼。您也可以指定表達式。

propertyValue

包含屬性值詳細資訊的結構。

quality

（選用）資產屬性值的品質。值必須為 GOOD、BAD 或 UNCERTAIN。您也可以指定表達式。

timestamp

（選用）包含時間戳記資訊的結構。如果您未指定此值，則預設為事件時間。

timeInSeconds

時間戳記 (以秒為單位，格式為 Unix epoch)。有效範圍介於 1 與 31556889864403199 之間。您也可以指定表達式。

offsetInNanos

（選用）從 轉換的奈秒位移`timeInSeconds`。有效範圍介於 0 與 999999999 之間。您也可以指定表達式。

value

包含資產屬性值的結構。

> **Important**
>
> 您必須根據 指定資產屬性的 `dataType`，指定下列其中一種值類型。如需詳細資訊，請參閱 AWS IoT SiteWise API 參考中的 AssetProperty。

booleanValue

（選用）資產屬性值是布林值，必須為 TRUE或 FALSE。您也可以指定表達式。如果您使用表達式，則評估結果應為布林值。

doubleValue

（選用）資產屬性值為兩倍。您也可以指定表達式。如果您使用表達式，評估結果應該是雙精確度。

integerValue

（選用）資產屬性值為整數。您也可以指定表達式。如果您使用表達式，則評估結果應該是整數。

stringValue

（選用）資產屬性值為字串。您也可以指定表達式。如果您使用表達式，則評估結果應該是字串。

> **Note**
>
> 請確定連接至 AWS IoT Events 服務角色的政策授予 `iotsitewise:BatchPutAssetPropertyValue`許可。如需詳細資訊，請參閱的身分和存取管理 AWS IoT Events。

如需詳細資訊，請參閱 AWS IoT Events API 參考中的 IotSiteWiseAction。

Amazon DynamoDB

DynamoDB action

Amazon DynamoDB 動作可讓您將資料傳送至 DynamoDB 資料表。DynamoDB 資料表的一個資料欄會在您指定的動作承載中接收所有屬性值對。如需支援的區域清單，請參閱《》中的 Amazon DynamoDB 端點和配額Amazon Web Services 一般參考。

Amazon DynamoDB 是一項完全受管的 NoSQL 資料庫服務，可提供快速且可預期的效能及無縫的可擴展性。如需詳細資訊，請參閱《Amazon DynamoDB 開發人員指南》中的什麼是 DynamoDB？。 DynamoDB

More information (10)

當您將資料傳送至 DynamoDB 資料表的一個資料欄時，您必須指定下列參數。

tableName

接收資料的 DynamoDB 資料表名稱。tableName 值必須符合 DynamoDB 資料表的資料表名稱。您也可以指定表達式。

hashKeyField

雜湊金鑰的名稱（也稱為分割區金鑰）。hashKeyField 值必須符合 DynamoDB 資料表的分割區索引鍵。您也可以指定表達式。

hashKeyType

（選用）雜湊金鑰的資料類型。雜湊金鑰類型的值必須為 STRING或 NUMBER。預設值為 STRING。您也可以指定表達式。

hashKeyValue

雜湊索引鍵的值。hashKeyValue 使用替代範本。這些範本會在執行時間提供資料。您也可以指定表達式。

rangeKeyField

(選用) 範圍索引鍵 (亦稱為排序索引鍵) 的名稱。rangeKeyField 值必須符合 DynamoDB 資料表的排序索引鍵。您也可以指定表達式。

rangeKeyType

（選用）範圍索引鍵的資料類型。雜湊金鑰類型的值必須為 STRING或 NUMBER。預設值為 STRING。您也可以指定表達式。

rangeKeyValue

(選用) 範圍索引鍵的值。rangeKeyValue 使用替代範本。這些範本會在執行時間提供資料。您也可以指定表達式。

operation

（選用）要執行的操作類型。您也可以指定表達式。操作值必須是下列其中一個值：

- INSERT - 將資料做為新項目插入 DynamoDB 資料表。這是預設值。
- UPDATE - 使用新資料更新 DynamoDB 資料表的現有項目。
- DELETE - 從 DynamoDB 資料表刪除現有項目。

`payloadField`

（選用）接收動作承載的 DynamoDB 資料欄名稱。預設名稱為 `payload`。您也可以指定表達式。

`payload`

（選用）預設承載包含具有偵測器模型執行個體和事件觸發動作相關資訊的所有屬性值對。您也可以自訂承載。如需詳細資訊，請參閱 AWS IoT Events API 參考中的承載。

如果指定的承載類型是字串，會將非 JSON 資料作為二進位資料`DynamoDBAction`傳送至 DynamoDB 資料表。DynamoDB 主控台會以 Base64 編碼文字來顯示資料。`payloadField` 值是 *`payload-field`*`_raw`。您也可以指定表達式。

> **Note**
>
> 請確定連接至 AWS IoT Events 服務角色的政策授予 `dynamodb:PutItem`許可。如需詳細資訊，請參閱的身分和存取管理 AWS IoT Events。

如需詳細資訊，請參閱 AWS IoT Events API 參考中的 DynamoDBAction。

Amazon DynamoDB(v2)

DynamoDBv2 action

Amazon DynamoDB(v2) 動作可讓您將資料寫入 DynamoDB 資料表。DynamoDB 資料表的個別資料欄會在您指定的動作承載中收到一個屬性值對。如需支援的區域清單，請參閱《》中的 Amazon DynamoDB 端點和配額Amazon Web Services 一般參考。

Amazon DynamoDB 是一項完全受管的 NoSQL 資料庫服務，可提供快速且可預期的效能及無縫的可擴展性。如需詳細資訊，請參閱《Amazon DynamoDB 開發人員指南》中的什麼是 DynamoDB？。 DynamoDB

More information (2)

當您將資料傳送至 DynamoDB 資料表的多個資料欄時，您必須指定下列參數。

`tableName`

接收資料的 DynamoDB 資料表名稱。您也可以指定表達式。

payload

（選用）預設承載包含具有偵測器模型執行個體和事件觸發動作相關資訊的所有屬性值對。您也可以自訂承載。如需詳細資訊，請參閱 AWS IoT Events API 參考中的承載。

> **Important**
>
> 承載類型必須是 JSON。您也可以指定表達式。

> **Note**
>
> 請確定連接至 AWS IoT Events 服務角色的政策授予 `dynamodb:PutItem`許可。如需詳細資訊，請參閱的身分和存取管理 AWS IoT Events。

如需詳細資訊，請參閱 AWS IoT Events API 參考中的 DynamoDBv2Action。

Amazon Data Firehose

Firehose action

Amazon Data Firehose 動作可讓您將資料傳送至 Firehose 交付串流。如需支援的區域清單，請參閱《》中的 Amazon Data Firehose 端點和配額Amazon Web Services 一般參考。

Amazon Data Firehose 是一項全受管服務，可將即時串流資料交付至 Amazon Simple Storage Service (Amazon Simple Storage Service)、Amazon Redshift、Amazon OpenSearch Service (OpenSearch Service) 和 Splunk 等目的地。如需詳細資訊，請參閱《Amazon Data Firehose 開發人員指南》中的什麼是 Amazon Data Firehose？。

More information (3)

當您將資料傳送至 Firehose 交付串流時，您必須指定下列參數。

deliveryStreamName

接收資料的 Firehose 交付串流名稱。

separator

（選用）您可以使用字元分隔符號來分隔傳送至 Firehose 交付串流的連續資料。分隔符號值必須是 '\n'（換行）、 '\t'(tab)、 '\r\n'(Windows 換行）或 ','（逗號）。

payload

（選用）預設承載包含具有偵測器模型執行個體和事件觸發動作相關資訊的所有屬性值對。您也可以自訂承載。如需詳細資訊，請參閱 AWS IoT Events API 參考中的承載。

> **Note**
>
> 請確定連接至 AWS IoT Events 服務角色的政策授予 `firehose:PutRecord`許可。如需詳細資訊，請參閱的身分和存取管理 AWS IoT Events。

如需詳細資訊，請參閱 AWS IoT Events API 參考中的 FirehoseAction。

AWS Lambda

Lambda action

AWS Lambda 動作可讓您呼叫 Lambda 函數。如需支援區域的清單，請參閱《Amazon Web Services 一般參考》中的 AWS Lambda 端點和配額。

AWS Lambda 是一種運算服務，可讓您執行程式碼，而無需佈建或管理伺服器。如需詳細資訊，請參閱《 AWS Lambda 開發人員指南》中的什麼是 AWS Lambda？。

More information (2)

當您呼叫 Lambda 函數時，您必須指定下列參數。

functionArn

Lambda 函數呼叫的 ARN。

payload

（選用）預設承載包含具有偵測器模型執行個體和事件觸發動作相關資訊的所有屬性值對。您也可以自訂承載。如需詳細資訊，請參閱 AWS IoT Events API 參考中的承載。

> **Note**
>
> 請確定連接至 AWS IoT Events 服務角色的政策授予 `lambda:InvokeFunction`許可。如需詳細資訊，請參閱的身分和存取管理 AWS IoT Events。

如需詳細資訊，請參閱 AWS IoT Events API 參考中的 LambdaAction。

Amazon Simple Notification Service

SNS action

Amazon SNS 主題發佈動作可讓您發佈 Amazon SNS 訊息。如需支援的區域清單，請參閱《》中的 Amazon Simple Notification Service 端點和配額Amazon Web Services 一般參考。

Amazon Simple Notification Service (Amazon Simple Notification Service) 是一種 Web 服務，可協調和管理訊息傳遞或傳送至訂閱端點或用戶端。如需詳細資訊，請參閱《Amazon Simple Notification Service 開發人員指南》中的什麼是 Amazon SNS？。

> **Note**
>
> Amazon SNS 主題發佈動作不支援 Amazon SNS FIFO （先進先出）主題。由於規則引擎是全分散式服務，因此啟動 Amazon SNS 動作時，訊息可能不會以指定的順序顯示。

More information (2)

當您發佈 Amazon SNS 訊息時，您必須指定下列參數。

`targetArn`

接收訊息之 Amazon SNS 目標的 ARN。

`payload`

（選用）預設承載包含具有偵測器模型執行個體和事件觸發動作相關資訊的所有屬性值對。您也可以自訂承載。如需詳細資訊，請參閱 AWS IoT Events API 參考中的承載。

> **Note**
>
> 請確定連接至 AWS IoT Events 服務角色的政策授予 `sns:Publish`許可。如需詳細資訊，請參閱的身分和存取管理 AWS IoT Events。

如需詳細資訊，請參閱《 AWS IoT Events API 參考》中的 SNSTopicPublishAction。

Amazon Simple Queue Service

SQS action

Amazon SQS 動作可讓您將資料傳送至 Amazon SQS 佇列。如需支援的區域清單，請參閱《》中的 Amazon Simple Queue Service 端點和配額Amazon Web Services 一般參考。

Amazon Simple Queue Service (Amazon SQS) 提供安全、耐用且可用的託管佇列，可讓您整合與分離分散式軟體系統和元件。如需詳細資訊，請參閱《Amazon Simple Queue Service 開發人員指南》中的什麼是 Amazon Simple Queue Service>。

> **Note**
>
> Amazon SQS 動作不支援 >Amazon SQS FIFO （先進先出）主題。由於規則引擎是全分散式服務，因此啟動 Amazon SQS 動作時，訊息可能不會以指定的順序顯示。

More information (3)

當您將資料傳送至 Amazon SQS 佇列時，您必須指定下列參數。

`queueUrl`

接收資料的 Amazon SQS 佇列 URL。

`useBase64`

如果您指定 ，（選用）會將資料 AWS IoT Events 編碼為 Base64 文字`TRUE`。預設值為 `FALSE`。

`payload`

（選用）預設承載包含具有偵測器模型執行個體和事件觸發動作相關資訊的所有屬性值對。您也可以自訂承載。如需詳細資訊，請參閱 AWS IoT Events API 參考中的承載。

> **Note**
>
> 請確定連接至 AWS IoT Events 服務角色的政策授予 `sqs:SendMessage`許可。如需詳細資訊，請參閱的身分和存取管理 AWS IoT Events。

如需詳細資訊，請參閱《 AWS IoT Events API 參考》中的 SNSTopicPublishAction。

您也可以使用 Amazon SNS 和 AWS IoT Core 規則引擎來觸發 AWS Lambda 函數。這可讓您使用其他 服務採取動作，例如 Amazon Connect，甚至是公司企業資源規劃 (ERP) 應用程式。

> **Note**
>
> 若要即時收集和處理大型資料記錄串流，您可以使用其他服務 AWS ，例如 Amazon Kinesis。從那裡，您可以完成初始分析，然後將結果作為偵測器的 AWS IoT Events 輸入傳送到 。

篩選、轉換和處理事件資料的表達式

運算式用於評估傳入資料、執行計算，以及判斷應發生特定動作或狀態轉換的條件。當您建立和更新偵測器模型時， AWS IoT Events 提供多種方式來指定值。您可以使用表達式來指定常值，也可以在指定特定值之前 AWS IoT Events 評估表達式。

主題

在 中篩選裝置資料和定義動作的語法 AWS IoT Events

運算式提供用於篩選裝置資料和定義動作的語法。您可以在表達式中使用 AWS IoT Events 常值、運算子、函數、參考和替換範本。透過結合這些元件，您可以建立強大且靈活的表達式來處理 IoT 資料、執行計算、操作字串，以及在偵測器模型中做出邏輯決策。

文字

- Integer
- Decimal (小數)
- String
- Boolean

運算子

Unary

- 非 （ 布林值 ）： !
- 非 （ 位元 ）： ~
- 最小值 （ 算術 ）： -

String

- 串連： **+**

  兩個運算元都必須是字串。字串常值必須以單引號 (') 括住。

例如：'my' + 'string'> 'mystring'

算術

- 新增： **+**

  兩個運算元都必須是數值。

- 減法： **-**
- 部門： **/**

  除非至少一個運算元（除數或除數）是十進位值，否則分割的結果是四捨五入整數值。

- 乘法： *****

位元（整數）

- OR： **|**

  例如：13 | 5> 13

- AND： **&**

  例如：13 & 5> 5

- XOR： **^**

  例如：13 ^ 5> 8

- 不適用： **~**

  例如：~13> -14

Boolean

- 小於： **<**
- 小於或等於： **<=**
- 等於： **==**
- 不等於： **!=**
- 大於或等於： **>=**
- 大於： **>**
- AND： **&&**
- OR： **||**

> **Note**
>
> 當 的子表達式||包含未定義的資料時，該子表達式會視為 false。

括號

您可以使用括號來分組表達式中的詞彙。

要在 AWS IoT Events 表達式中使用的函數

AWS IoT Events 提供一組內建函數，以增強偵測器模型表達式的功能。這些函數可啟用計時器管理、類型轉換、Null 檢查、觸發類型識別、輸入驗證、字串操作和位元操作。透過利用這些函數，您可以建立回應式 AWS IoT Events 處理邏輯，改善 IoT 應用程式的整體效率。

內建函數

`timeout("timer-name")`

`true` 如果指定的計時器已過，則評估 為 。將 "*timer-name*" 取代為您定義的計時器名稱，以引號表示。在事件動作中，您可以定義計時器，然後啟動計時器、重設計時器，或清除先前定義的計時器。請參閱 欄位 `detectorModelDefinition.states.onInput|onEnter|onExit.events.actions.setTimer.timerName`。

您可以在不同狀態中參考處於某個狀態的計時器集。您必須先造訪您建立計時器的狀態，才能進入參考計時器的狀態。

例如，偵測器模型有兩個狀態 `TemperatureChecked`和 `RecordUpdated`。您已在 `TemperatureChecked` 狀態建立計時器。您必須先造訪 `TemperatureChecked` 狀態，才能使用 `RecordUpdated` 狀態的計時器。

為了確保準確性，應設定計時器的最短時間為 60 秒。

> **Note**
>
> `timeout()` `true`只會在實際計時器過期後第一次檢查時傳回 ，並在`false`之後傳回 。

convert(*type*, *expression*)

評估為轉換為指定類型的表達式值。*##*值必須為 String、 Boolean或 Decimal。使用其中一個關鍵字或評估為包含關鍵字之字串的表達式。只有下列轉換成功並傳回有效值：

- 布林值 -> 字串

  傳回字串 "true"或 "false"。

- 小數 -> 字串
- 字串 -> 布林值
- 字串 -> 小數

  指定的字串必須是小數的有效表示法，否則convert()會失敗。

如果 convert()未傳回有效的值，則其所屬的表達式也會無效。此結果等同於 false ，且不會觸發 actions或 轉換為nextState指定為發生表達式之事件的一部分。

isNull(*expression*)

true 如果表達式傳回 null，則評估為 。例如，如果輸入MyInput收到訊息 { "a": null }，則以下內容會評估為 true，但isUndefined($input.MyInput.a)評估為 false。

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

isUndefined(*expression*)

true 如果未定義表達式，則評估為 。例如，如果輸入MyInput收到訊息 { "a": null }，則下列項目會評估為 false，但isNull($input.MyInput.a)評估為 true。

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

triggerType("*type*")

*##*值可以是 "Message"或 "Timer"。評估 true 是否因為計時器過期而顯示的事件條件正在評估，如下列範例所示。

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

或者收到輸入訊息。

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

currentInput("*input*")

評估 true是否因為收到指定的輸入訊息而評估其出現的事件條件。例如，如果輸入Command收到訊息 { "value": "Abort" }，則以下內容會評估為 true。

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

使用此函數來驗證正在評估條件，因為已收到特定輸入且計時器尚未過期，如下列表達式所示。

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

字串比對函數

startsWith(*expression1*, *expression2*)

評估第一個字串表達式true是否以第二個字串表達式開頭。例如，如果輸入MyInput收到訊息 { "status": "offline"}，則以下內容會評估為 true。

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

兩個表達式都必須評估為字串值。如果任一表達式未評估為字串值，則函數的結果為未定義。不會執行轉換。

endsWith(*expression1*, *expression2*)

true 如果第一個字串表達式以第二個字串表達式結尾，則評估為 。例如，如果輸入MyInput收到訊息 { "status": "offline" }，則以下內容會評估為 true。

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

兩個表達式都必須評估為字串值。如果任一表達式未評估為字串值，則函數的結果為未定義。不會執行轉換。

contains(*expression1*, *expression2*)

評估第一個字串表達式true是否包含第二個字串表達式。例如，如果輸入MyInput收到訊息 { "status": "offline" }，則以下內容會評估為 true。

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

兩個表達式都必須評估為字串值。如果任一表達式未評估為字串值，則函數的結果為未定義。不會執行轉換。

Bitwise 整數控制函數

bitor(*expression1*, *expression2*)

評估整數表達式的位元 OR（二進位 OR 操作會在整數的對應位元上執行）。例如，如果輸入MyInput收到訊息 `{ "value1": 13, "value2": 5 }`，則以下內容會評估為 13。

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

兩個表達式都必須評估為整數值。如果任一表達式未評估為整數值，則函數的結果為未定義。不會執行轉換。

bitand(*expression1*, *expression2*)

評估整數表達式的位元 AND（二進位 AND 操作會在整數的對應位元上執行）。例如，如果輸入MyInput收到訊息 `{ "value1": 13, "value2": 5 }`，則以下內容會評估為 5。

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

兩個表達式都必須評估為整數值。如果任一表達式未評估為整數值，則函數的結果為未定義。不會執行轉換。

bitxor(*expression1*, *expression2*)

評估整數表達式的位元 XOR（二進位 XOR 操作會在整數的對應位元上執行）。例如，如果輸入MyInput收到訊息 `{ "value1": 13, "value2": 5 }`，則以下內容會評估為 8。

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

兩個表達式都必須評估為整數值。如果任一表達式未評估為整數值，則函數的結果為未定義。不會執行轉換。

bitnot(*expression*)

評估整數表達式的位元 NOT（二進位 NOT 操作是在整數的位元上執行）。例如，如果輸入MyInput收到訊息 `{ "value": 13 }`，則以下內容會評估為 -14。

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

兩個表達式都必須評估為整數值。如果任一表達式未評估為整數值，則函數的結果為未定義。不會執行轉換。

AWS IoT Events 運算式中輸入和變數的參考

輸入

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

input-name 是您使用 CreateInput 動作建立的輸入。

例如，如果您有名為 的輸入TemperatureInput來定義inputDefinition.attributes.jsonPath項目，則值可能會出現在下列可用欄位中。

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

若要參考 temperature 欄位的值，請使用下列命令。

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

對於值為陣列的欄位，您可以使用 來參考陣列的成員[*n*]。例如，指定下列值：

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

您可以使用下列命令78.8參考 值。

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

Variables

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

`variable-name` 是您使用 CreateDetectorModel 動作定義的變數。

例如，如果您有使用 TechnicianID 定義的名為 的變數detectorDefinition.states.onInputEvents.actions.setVariable.variableName，您可以使用下列命令參考最近提供給變數的 （ 字串 ） 值。

```
$variable.TechnicianID
```

您只能使用 setVariable動作來設定變數的值。您無法為表達式中的變數指派值。變數無法取消設定。例如，您無法為其指派值 null。

> **Note**
>
> 在使用不遵循 （ 規則表達式 ） 模式的識別符的參考中[a-zA-Z][a-zA-Z0-9_]*，您必須在反引號 () 中括住這些識別符`。例如，對於名為 MyInput且具有名為 欄位的輸入參考_value，必須將此欄位指定為 $input.MyInput.`_value`。

當您在表達式中使用參考時，請檢查下列項目：

- 當您使用參考做為具有一或多個運算子的運算元時，請確定您參考的所有資料類型都相容。

  例如，在以下表達式中，整數2是 ==和 &&運算子的運算元。為了確保運算元相容，$variable.testVariable + 1且$variable.testVariable必須參考整數或小數。

  此外，整數1是+運算子的運算元。因此， $variable.testVariable 必須參考整數或小數。

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

- 當您使用參考做為傳遞給函數的引數時，請確定該函數支援您參考的資料類型。

  例如，下列timeout("*time-name*")函數需要具有雙引號的字串做為引數。如果您使用#####值的參考，則必須參考具有雙引號的字串。

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

> **Note**
>
> 對於 convert(*type*, *expression*)函數，如果您使用##值的參考，則參考的評估結果必須是 String、 Decimal或 Boolean。

AWS IoT Events 運算式支援整數、小數、字串和布林資料類型。下表提供不相容的類型對清單。

類型配對不相容
整數、字串
整數，布林值
小數，字串
小數，布林值
字串，布林值

AWS IoT Events 表達式的替代範本

'${*expression*}'

會將字串${}識別為插入字串。expression 可以是任何 AWS IoT Events 表達式。這包括運算子、函數和參考。

例如，您使用 SetVariableAction 動作來定義變數。variableName 即為 SensorID，而 value 即為 10。您可以建立下列替代範本。

替代範本	結果字串
`'${'Sensor ' + $variable.SensorID}'`	"Sensor 10"
`'Sensor ' + '${$variable.SensorID + 1}'`	"Sensor 11"

替代範本	結果字串
`'Sensor 10: ${$variable.SensorID == 10}'`	`"Sensor 10: true"`
`'{\"sensor\":\"${$variable.SensorID + 1}\"}'`	`"{\"sensor\":\"11\"}"`
`'{\"sensor\":${$variable.SensorID + 1}}'`	`"{\"sensor\":11}"`

的表達式範例和用量 AWS IoT Events

您可以透過下列方式在偵測器模型中指定值：

- 在 AWS IoT Events 主控台中輸入支援的表達式。
- 將表達式傳遞至 AWS IoT Events APIs做為參數。

運算式支援常值、運算子、函數、參考和替代範本。

> **Important**
>
> 您的表達式必須參考整數、小數、字串或布林值。

撰寫 AWS IoT Events 表達式

請參閱下列範例，協助您撰寫 AWS IoT Events 表達式：

常值

對於文字值，表達式必須包含單引號。布林值必須為 `true`或 `false`。

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

```
'true'       # Boolean
```

參考資料

對於參考，您必須指定變數或輸入值。

- 下列輸入參考十進位數字 10.01。

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

- 下列變數參考字串 Greenhouse Temperature Table。

```
$variable.TableName
```

替代範本

對於替代範本，您必須使用 ${}，且範本必須以單引號括起來。替代範本也可以包含文字、運算子、函數、參考和替代範本的組合。

- 下列表達式的評估結果是字串 50.018 in Fahrenheit。

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

- 下列表達式的評估結果是字串 {\"sensor_id\":\"Sensor_1\",\"temperature\": \"50.018\"}。

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

字串串連

對於字串串連，您必須使用 +。字串串連也可以包含文字、運算子、函數、參考和替代範本的組合。

- 下列表達式的評估結果是字串 Greenhouse Temperature Table 2000-01-01。

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

AWS IoT Events 偵測器模型範例

此頁面提供範例使用案例的清單，示範如何設定各種 AWS IoT Events 功能。範例範圍從基本偵測，例如溫度閾值，到更進階的異常偵測和機器學習案例。每個範例都包含程序和程式碼片段，協助您設定 AWS IoT Events 偵測、動作和整合。這些範例展示 AWS IoT Events 服務的彈性，以及如何針對各種 IoT 應用程式和使用案例進行自訂。探索 AWS IoT Events 功能時或如果您需要實作特定偵測或自動化工作流程的指引，請參閱此頁面。

主題

範例：搭配 使用 HVAC 溫度控制 AWS IoT Events

背景故事

此範例實作具有下列功能的溫度控制模型（調溫器）：

- 您定義的一個偵測器模型，可監控和控制多個區域。（將為每個區域建立偵測器執行個體。）
- 每個偵測器執行個體會從每個控制區域中放置的多個感應器接收溫度資料。
- 您可以隨時變更每個區域的所需溫度（設定點）。
- 您可以定義每個區域的操作參數，並隨時變更這些參數。
- 您可以隨時從 區域新增或刪除感應器。
- 您可以針對時間加熱和冷卻單元啟用最低執行，以保護它們免於損壞。
- 偵測器會拒絕並報告異常的感應器讀數。

- 您可以定義緊急溫度設定點。如果任何一個感應器報告高於或低於您定義之設定點的溫度，則會立即接合加熱或冷卻單元，而偵測器會報告該溫度峰值。

此範例示範下列功能：

- 建立事件偵測器模型。
- 建立輸入。
- 將輸入擷取至偵測器模型。
- 評估觸發條件。
- 請參閱條件中的狀態變數，並根據條件設定變數的值。
- 請參閱條件中的計時器，並根據條件設定計時器。
- 採取傳送 Amazon SNS 和 MQTT 訊息的動作。

中的 HVAC 系統的輸入定義 AWS IoT Events

`seedTemperatureInput` 用於建立區域的偵測器執行個體，並定義其操作參數。

在 中設定 HVAC 系統的輸入對於有效的氣候控制 AWS IoT Events 至關重要。此範例說明如何設定擷取參數的輸入，例如溫度、濕度、佔用率和能源消耗資料。了解如何定義輸入屬性、設定資料來源和設定預先處理規則，以協助偵測器模型接收準確且及時的資訊，以獲得最佳管理和效率。

使用的 CLI 命令：

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

檔案： `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" }
    ]
  }
}
```

回應：

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

應視需要由每個區域中的每個感應器temperatureInput傳送 。

使用的 CLI 命令：

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

檔案： temperatureInput.json

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

回應：

```
{
    "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 的 HVAC 系統的偵測器模型定義 AWS IoT Events

areaDetectorModel 定義每個偵測器執行個體的運作方式。每個state machine執行個體都會擷取溫度感應器讀數，然後根據這些讀數變更狀態並傳送控制訊息。

使用的 CLI 命令：

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

檔案： 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"
}
```

回應：

```
{
    "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"
    }
}
```

中的 HVAC 系統的 BatchPutMessage 範例 AWS IoT Events

在此範例中， BatchPutMessage 用於為 區域建立偵測器執行個體，並定義初始操作參數。

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
```

檔案： 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}"
    }
  ]
}
```

回應：

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

在此範例中， BatchPutMessage 用於報告 區域中單一感應器的溫度感應器讀數。

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
```

檔案： temperatureExample.json

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

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

回應：

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

在此範例中，BatchPutMessage 用於變更 區域的所需溫度。

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
```

檔案：seedSetDesiredTemp.json

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

回應：

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

Area51 偵測器執行個體產生的 Amazon SNS 訊息範例：

```
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"
}
```

在此範例中，我們使用 DescribeDetector API 取得偵測器執行個體目前狀態的相關資訊。

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

回應：

```
{
    "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"
    }
}
```

中的 HVAC 系統的 BatchUpdateDetector 範例 AWS IoT Events

在此範例中， `BatchUpdateDetector` 用於變更運作中偵測器執行個體的操作參數。

高效的 HVAC 系統管理通常需要批次更新多個偵測器。本節示範如何使用偵測器的 AWS IoT Events批次更新功能。了解如何同時修改多個控制參數、更新閾值，以便您可以調整裝置機群的回應動作，從而提高有效管理大規模系統的能力。

使用的 CLI 命令：

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

檔案： 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": [
        ]
      }
    }
  ]
}
```

回應：

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

AWS IoT Core 規則引擎和 AWS IoT Events

下列規則會將 AWS IoT Events MQTT 訊息重新發佈為影子更新請求訊息。我們假設為由偵測器模型控制的每個區域定義了供暖單位和冷卻單位的 AWS IoT Core 物件。

在此範例中，我們定義了名為 `Area51HeatingUnit`和 的物件`Area51CoolingUnit`。

使用的 CLI 命令：

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

檔案：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"
        }
      }
    ]
  }
}
```

回應：【空白】

使用的 CLI 命令：

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

檔案：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"
        }
      }
    ]
  }
}
```

回應：【空白】

使用的 CLI 命令：

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

檔案：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"
        }
      }
    ]
  }
}
```

回應：【空白】

使用的 CLI 命令：

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

檔案： 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"
        }
      }
    ]
  }
}
```

回應：【空白】

範例：使用 偵測條件的 Cane AWS IoT Events

許多 Cane 的運算子想要偵測機器何時需要維護或替換，並觸發適當的通知。每個 Cane 都有一個馬達。馬達會發出訊息（輸入），其中包含壓力和溫度的相關資訊。運算子想要兩個層級的事件偵測器：

- 角層級事件偵測器
- 馬達層級事件偵測器

使用來自馬達的訊息（包含具有 craneId和 的中繼資料motorid)，運算子可以使用適當的路由來執行兩個層級的事件偵測器。符合事件條件時，通知應傳送至適當的 Amazon SNS 主題。操作員可以設定偵測器模型，以免引發重複的通知。

此範例示範下列功能：

- 建立、讀取、更新、刪除 (CRUD) 輸入。
- 建立、讀取、更新、刪除 (CRUD) 事件偵測器模型和不同版本的事件偵測器。
- 將一個輸入路由到多個事件偵測器。
- 將輸入擷取至偵測器模型。
- 觸發條件和生命週期事件的評估。
- 能夠參考條件中的狀態變數，並根據條件設定其值。
- 具有定義、狀態、觸發器評估器和動作執行器的執行時間協同運作。
- `ActionsExecutor` 使用 SNS 目標在 中執行動作。

傳送命令以回應 中偵測到的條件 AWS IoT Events

此頁面提供使用 AWS IoT Events 命令來設定輸入、建立偵測器模型和傳送模擬感應器資料的範例。這些範例示範如何利用 AWS IoT Events 來監控 馬達和 crane 等工業設備。

```
#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
```

用於 Cane 監控的 AWS IoT Events 偵測器模型

監控您的設備或裝置機群是否有操作失敗或變更，並在發生此類事件時觸發動作。您可以在指定狀態、規則和動作的 JSON 中定義偵測器模型。這可讓您監控輸入，例如溫度和壓力、追蹤閾值違規，以及傳送提醒。這些範例顯示機群和馬達的偵測器模型，偵測過熱問題，並在超過閾值時由 Amazon SNS 通知。您可以更新模型來精簡行為，而不會中斷監控。

檔案： `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"
}
```

更新現有的偵測器模型。檔案： 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"
}
```

檔案：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"
}
```

更新現有的偵測器模型。檔案： 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 用於 Cane 監控的 輸入

在這個範例中，我們會示範如何使用 來設定 Cane 監控系統的輸入 AWS IoT Events。它會擷取壓力和溫度輸入，以說明如何為複雜的工業設備監控建構輸入。

檔案： `pressureInput.json`

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

檔案： `temperatureInput.json`

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

使用 傳送警示和操作訊息 AWS IoT Events

有效的訊息處理在 Cane 監控系統中很重要。本節示範如何設定 AWS IoT Events 以處理和回應來自 crane 感應器的各種訊息類型。根據特定訊息設定警示可協助您剖析、篩選和路由狀態更新，以觸發適當的動作。

檔案： highPressureMessage.json

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

        }
    ]
}
```

檔案： highTemperatureMessage.json

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

檔案： lowPressureMessage.json

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

```
        }
    ]
}
```

檔案：lowTemperatureMessage.json

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

範例：使用感應器和應用程式 AWS IoT Events 進行事件偵測

此偵測器模型是 AWS IoT Events 主控台提供的其中一個範本。為方便起見，它包含在這裡。

此範例示範使用感應器資料偵測 AWS IoT Events應用程式事件。它顯示如何建立監控指定事件的偵測器模型，以便您可以觸發適當的動作。您可以建立多個感應器輸入、定義複雜的事件條件，以及設定漸進式回應機制。

```
{
    "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"
    }
}
```

範例：使用 Device HeartBeat 監控裝置與 的連線 AWS IoT Events

此偵測器模型是 AWS IoT Events 主控台提供的其中一個範本。為方便起見，它包含在這裡。

瑕疵心跳 (DHB) 範例說明如何 AWS IoT Events 在醫療保健監控中使用 。此範例示範如何建立偵測器模型，以分析心率資料、偵測不規則模式，並觸發適當的回應。了解如何設定輸入、定義閾值，以及設定潛在心臟問題的提醒，展示相關醫療保健應用程式的 AWS IoT Events多樣性。

```
{
    "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"
    }
}
```

範例： 中的 ISA 警示 AWS IoT Events

此偵測器模型是 AWS IoT Events 主控台提供的其中一個範本。為方便起見，它包含在這裡。

```
{
    "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"
}
```

範例：使用 建置簡單的警示 AWS IoT Events

此偵測器模型是 AWS IoT Events 主控台提供的其中一個範本。為方便起見，它包含在這裡。

```
{
    "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"
}
```

在 中使用警示進行監控 AWS IoT Events

AWS IoT Events 警示可協助您監控資料是否有變更。資料可以是您為設備和程序測量的指標。您可以建立警示，在超過閾值時傳送通知。警示可協助您偵測問題、簡化維護，以及最佳化設備和程序的效能。

警示是警示模型的執行個體。警示模型會指定要偵測的內容、何時傳送通知、誰會收到通知等。您也可以指定警示狀態變更時發生的一或多個支援動作。 AWS IoT Events 將衍生自您資料的輸入屬性路由至適當的警示。如果您監控的資料超出指定範圍，則會叫用警示。您也可以確認警示，或將其設定為休眠模式。

使用 AWS IoT SiteWise

您可以使用 AWS IoT Events 警示來監控資產屬性 in AWS IoT SiteWise. AWS IoT SiteWise sends 資產屬性值到 AWS IoT Events alarm. AWS IoT Events send the alarm status to AWS IoT SiteWise。

AWS IoT SiteWise 也支援外部警示。如果您在 外部使用警示， AWS IoT SiteWise 並擁有傳回警示狀態資料的解決方案，您可以選擇外部警示。外部警示包含擷取警示狀態資料的測量屬性。

AWS IoT SiteWise 不會評估外部警示的狀態。此外，您無法在警示狀態變更時確認或休眠外部警示。

您可以使用 SiteWise Monitor 功能，在 SiteWise Monitor 入口網站中檢視外部警示的狀態。

如需詳細資訊，請參閱AWS IoT SiteWise 《 使用者指南》中的使用警示監控資料，以及《 SiteWise Monitor 應用程式指南》中的使用警示監控資料。

確認流程

當您建立警示模型時，您可以選擇是否要啟用確認流程。如果您啟用確認流程，您的團隊會在警示狀態變更時收到通知。您的團隊可以確認警示並留下備註。例如，您可以包含警示的資訊，以及您要採取的動作來解決問題。如果您監控的資料超出指定範圍，則會叫用警示。

警示具有下列狀態：

`DISABLED`

當警示處於 `DISABLED` 狀態時，尚未準備好評估資料。若要啟用警示，您必須將警示變更為 `NORMAL` 狀態。

NORMAL

當警示處於 NORMAL 狀態時，即可評估資料。

ACTIVE

如果警示處於 ACTIVE 狀態，則會叫用警示。您正在監控的資料超出指定範圍。

ACKNOWLEDGED

當警示處於 ACKNOWLEDGED 狀態時，會叫用警示，而且您確認警示。

LATCHED

警示已調用，但您在一段時間後未確認警示。警示會自動變更為 NORMAL 狀態。

SNOOZE_DISABLED

當警示處於 SNOOZE_DISABLED 狀態時，警示會在指定的期間內停用。在休眠時間之後，警示會自動變更為 NORMAL 狀態。

在 中建立警示模型 AWS IoT Events

您可以使用 AWS IoT Events 警示來監控您的資料，並在超過閾值時收到通知。警示提供您用來建立或設定警示模型的參數。您可以使用 AWS IoT Events 主控台或 AWS IoT Events API 來建立或設定警示模型。當您設定警示模型時，變更會在新資料送達時生效。

要求

當您建立警示模型時，適用下列要求。

- 您可以建立警示模型來監控 中的輸入屬性 AWS IoT Events 或 中的資產屬性 AWS IoT SiteWise。
  - 如果您選擇在建立警示模型 AWS IoT Events在 中建立模型的輸入 AWS IoT Events之前監控 中的輸入屬性。
  - 如果您選擇監控資產屬性，您必須先在 中建立資產模型， AWS IoT SiteWise 才能建立警示模型。
- 您必須擁有允許警示執行動作和存取 AWS 資源的 IAM 角色。如需詳細資訊，請參閱設定 的許可 AWS IoT Events。
- 本教學課程使用的所有 AWS 資源都必須位於相同的 AWS 區域。

建立警示模型（主控台）

以下說明如何在 AWS IoT Events 主控台中建立警示模型來監控 AWS IoT Events 屬性。

1. 登入 AWS IoT Events 主控台。
2. 在導覽窗格中，選擇警示模型。
3. 在警示模型頁面上，選擇建立警示模型。
4. 在警示模型詳細資訊區段中，執行下列動作：
   a. 請輸入唯一的名稱。
   b. (選用) 輸入描述。
5. 在警示目標區段中，執行下列動作：

   > **Important**
   > 如果您選擇AWS IoT SiteWise 資產屬性，則必須已在其中建立資產模型 AWS IoT SiteWise。

   a. 選擇AWS IoT Events 輸入屬性。
   b. 選擇輸入。
   c. 選擇輸入屬性索引鍵。此輸入屬性用作建立與警示金鑰相關聯之 alarm. AWS IoT Events routes 輸入的金鑰。

      > **Important**
      > 如果輸入訊息承載不包含此輸入屬性金鑰，或如果金鑰不在金鑰中指定的相同 JSON 路徑中，則訊息將失敗擷取 AWS IoT Events。

6. 在閾值定義區段中，您可以定義 AWS IoT Events 用於變更警示狀態的輸入屬性、閾值和比較運算子。
   a. 針對輸入屬性，選擇您要監控的屬性。

      每次此輸入屬性收到新資料時，都會進行評估，以判斷警示的狀態。
   b. 針對運算子，選擇比較運算子。運算子會將您的輸入屬性與屬性的閾值進行比較。

您可以從這些選項中選擇：

- > 大於
- >= 大於或等於
- < 小於
- <= 小於或等於
- = 等於
- ！= 不等於

c. 對於閾值，在 AWS IoT Events input. AWS IoT Events compares 中輸入數字或選擇屬性。將此值與您選擇的輸入屬性值進行比較。

d. （選用）對於嚴重性，請使用您的團隊理解的數字來反映此警示的嚴重性。

7. （選用）在通知設定區段中，設定警示的通知設定。

您最多可以新增 10 個通知。對於通知 1，請執行下列動作：

a. 針對通訊協定，從下列選項中選擇：

- 電子郵件和文字 - 警示會傳送簡訊通知和電子郵件通知。
- 電子郵件 - 警示會傳送電子郵件通知。
- 文字 - 警示會傳送簡訊通知。

b. 針對寄件者，指定可傳送此警示通知的電子郵件地址。

若要將更多電子郵件地址新增至寄件者清單，請選擇新增寄件者。

c. （選用）針對收件人，選擇收件人。

若要將更多使用者新增至收件人清單，請選擇新增使用者。您必須先將新使用者新增至 IAM Identity Center 存放區，才能將其新增至警示模型。如需詳細資訊，請參閱在 中管理警示收件人的 IAM Identity Center 存取權 AWS IoT Events。

d. （選用）對於其他自訂訊息，輸入說明警示偵測到的內容以及收件人應採取的動作的訊息。

8. 在執行個體區段中，您可以啟用或停用根據此警示模型建立的所有警示執行個體。

9. 在進階設定區段中，執行下列動作：

a. 對於確認流程，您可以啟用或停用通知。

- 如果您選擇已啟用，您會在警示狀態變更時收到通知。您必須先確認通知，警示狀態才能恢復正常。
- 如果您選擇已停用，則不需要採取任何動作。當測量傳回指定範圍時，警示會自動變更為正常狀態。

如需詳細資訊，請參閱確認流程。

b. 針對許可，選擇下列其中一個選項：

- 您可以從 AWS 政策範本建立新的角色，並 AWS IoT Events 自動為您建立 IAM 角色。
- 您可以使用現有的 IAM 角色，允許此警示模型執行動作並存取其他 AWS 資源。

如需詳細資訊，請參閱適用於 AWS IoT Events的 Identity and Access Management。

c. 對於其他通知設定，您可以編輯 AWS Lambda 函數來管理警示通知。為您的 AWS Lambda 函數選擇下列其中一個選項：

- 建立新的 AWS Lambda 函數 - 為您 AWS IoT Events 建立新的 AWS Lambda 函數。
- 使用現有 AWS Lambda 函數 - 透過選擇 AWS Lambda AWS Lambda 函數名稱來使用現有函數。

如需可能動作的詳細資訊，請參閱 AWS IoT Events 使用其他 AWS 服務。

d. （選用）對於設定狀態動作，您可以新增警示狀態變更時要採取的一或多個 AWS IoT Events 動作。

10. （選用）您可以新增標籤來管理警示。如需詳細資訊，請參閱標記您的 AWS IoT Events 資源。
11. 選擇建立。

在 中回應警示 AWS IoT Events

有效回應警示是使用 管理 IoT 系統的重要層面 AWS IoT Events。探索設定和處理警示的各種方式，包括：設定通知管道、定義呈報程序，以及實作自動化回應動作。了解如何建立細微的警示條件、排定警示的優先順序，並與其他 AWS 服務整合，為您的 IoT 應用程式建置回應靈敏的警示管理系統。

如果您啟用確認流程，則會在警示狀態變更時收到通知。若要回應警示，您可以確認、停用、啟用、重設或暫停警示。

Console

以下說明如何在 主控台中 AWS IoT Events 回應警示。

1. 登入 AWS IoT Events 主控台。
2. 在導覽窗格中，選擇警示模型。
3. 選擇目標警示模型。
4. 在警示清單區段中，選擇目標警示。
5. 您可以從動作中選擇下列其中一個選項：

   - 確認 - 警示變更為 ACKNOWLEDGED 狀態。
   - 停用 - 警示會變更為 DISABLED 狀態。
   - 啟用 - 警示會變更為 NORMAL 狀態。
   - 重設 - 警示會變更為 NORMAL 狀態。
   - 暫停，然後執行下列動作：

     1. 選擇 Snooze 長度或輸入自訂 Snooze 長度。
     2. 選擇儲存。

     警示會變更為 SNOOZE_DISABLED 狀態

   如需警示狀態的詳細資訊，請參閱 確認流程。

API

若要回應一或多個警示，您可以使用下列 AWS IoT Events API 操作：

- BatchAcknowledgeAlarm
- BatchDisableAlarm
- BatchEnableAlarm
- BatchResetAlarm
- BatchSnoozeAlarm

在 中管理警示通知 AWS IoT Events

AWS IoT Events 與 Lambda 整合，提供自訂事件處理功能。本節說明如何在 AWS IoT Events 偵測器模型中使用 Lambda 函數，讓您執行複雜的邏輯、與外部服務互動，以及實作複雜的事件處理。

AWS IoT Events 使用 Lambda 函數來管理警示通知。您可以使用 提供的 Lambda 函數， AWS IoT Events 或建立新的函數。

主題

在 中建立 Lambda 函數 AWS IoT Events

AWS IoT Events 提供 Lambda 函數，可讓警示傳送和接收電子郵件和簡訊通知。

要求

當您為警示建立 Lambda 函數時，適用下列要求：

- 如果您的警示傳送簡訊通知，請確定 Amazon SNS 已設定為傳送簡訊。
  - 如需詳細資訊，請參閱下列 文件：
    - 《Amazon Simple Notification Service 開發人員指南》中的具有 Amazon SNS 和 Amazon SNS SMS 訊息來源身分的行動簡訊。
    - AWS SMS 《 使用者指南》中的什麼是 AWS 最終使用者傳訊 SMS？。
- 如果您的警示傳送電子郵件或簡訊通知，您必須具有允許 AWS Lambda 使用 Amazon SES 和 Amazon SNS 的 IAM 角色。

  範例政策：

  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" : "*"
        }
    ]
}
```

- 您必須為 AWS IoT Events 和 選擇相同的 AWS 區域 AWS Lambda。如需支援的區域清單，請參閱中的AWS IoT Events 端點和配額，以及AWS Lambda 端點和配額Amazon Web Services 一般參考。

部署 Lambda 函數以 AWS IoT Events 使用 CloudFormation

本教學課程使用 CloudFormation 範本來部署 Lambda 函數。此範本會自動建立 IAM 角色，允許 Lambda 函數使用 Amazon SES 和 Amazon SNS。

以下說明如何使用 AWS Command Line Interface (AWS CLI) 來建立 CloudFormation 堆疊。

1. 在裝置的終端機中，執行 `aws --version` 以檢查是否已安裝 AWS CLI。如需詳細資訊，請參閱 AWS Command Line Interface User Guide 中的 Installing or updating to the latest version of the AWS CLI。
2. 執行 `aws configure list` 以檢查您是否 AWS CLI 在具有本教學課程所有 AWS 資源的 AWS 區域中設定 。如需詳細資訊，請參閱AWS Command Line Interface 《 使用者指南》中的使用命令設定和檢視組態設定
3. 下載 CloudFormation 範本 notificationLambda.template.yaml.zip。

   > **Note**
   >
   > 如果您在下載檔案時遇到困難，也可以在 中使用範本CloudFormation 範本。

4. 解壓縮內容並以 `notificationLambda.template.yaml` 儲存在本機。
5. 在您的裝置上開啟終端機，並導覽至您下載`notificationLambda.template.yaml`檔案的目錄。
6. 若要建立 CloudFormation 堆疊，請執行下列命令：

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

您可以修改此 CloudFormation 範本來自訂 Lambda 函數及其行為。

> **Note**
>
> AWS Lambda 會重試函數錯誤兩次。如果函式沒有足夠的容量來處理所有傳入的請求，事件可能在佇列中等待數小時或數天才會傳送到函式。您可以在函數上設定未傳遞訊息佇列 (DLQ)，以擷取未成功處理的事件。如需詳細資訊，請參閱 AWS Lambda 開發人員指南中的非同步叫用。

您也可以在 CloudFormation 主控台中建立或設定堆疊。如需詳細資訊，請參閱AWS CloudFormation《 使用者指南》中的使用堆疊。

為 建立自訂 Lambda 函數 AWS IoT Events

您可以建立 Lambda 函數或修改 提供的函數 AWS IoT Events。

當您建立自訂 Lambda 函數時，適用下列要求。

- 新增允許 Lambda 函數執行指定動作和存取 AWS 資源的許可。
- 如果您使用 提供的 Lambda 函數 AWS IoT Events，請務必選擇 Python 3.7 執行時間。

Lambda 函數範例：

```
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')
```

如需詳細資訊，請參閱 AWS Lambda 開發人員指南中的什麼是 AWS Lambda？。

CloudFormation 範本

使用下列 CloudFormation 範本建立 Lambda 函數。

```
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 提供的 Lambda 函數 AWS IoT Events

透過警示通知，您可以使用 提供的 Lambda 函數 AWS IoT Events 來管理警示通知。

當您使用 AWS IoT Events 提供的 Lambda 函數來管理警示通知時，適用下列要求：

- 您必須驗證在 Amazon Simple Email Service (Amazon SES中傳送電子郵件通知的電子郵件地址。如需詳細資訊，請參閱《Amazon Simple Email Service 開發人員指南》中的驗證電子郵件地址身分。

  如果您收到驗證連結，請按一下連結來驗證您的電子郵件地址。您也可以檢查垃圾郵件資料夾是否有驗證電子郵件。
- 如果您的警示傳送簡訊通知，您必須針對電話號碼使用 E.164 國際電話號碼格式。此格式包含 `+<country-calling-code><area-code><phone-number>`。

  電話號碼範例：

Country	本機電話號碼	E.164 格式數字
美國	206-555-0100	+12065550100
英國	020-1234-1234	+442012341234
立陶宛	8+601+12345	+37060112345

若要尋找國家/地區呼叫代碼，請前往 https：//countrycode.org。

提供的 Lambda 函數會 AWS IoT Events 檢查您是否使用 E.164 格式的電話號碼。不過，它不會驗證電話號碼。如果您確定您輸入了正確的電話號碼，但未收到簡訊通知，您可以聯絡電信業者。電信業者可能會封鎖訊息。

在 中管理警示收件人的 IAM Identity Center 存取權 AWS IoT Events

AWS IoT Events 使用 AWS IAM Identity Center 來管理警示收件人的 SSO 存取。為 AWS IoT Events 通知收件人實作 IAM Identity Center 可以增強安全性和使用者體驗。若要啟用警示以傳送通知給收件人，您必須啟用 IAM Identity Center，並將收件人新增至您的 IAM Identity Center 存放區。如需詳細資訊，請參閱AWS IAM Identity Center 《 使用者指南》中的新增使用者。

> **Important**
>
> - 您必須為 AWS IoT Events AWS Lambda和 IAM Identity Center 選擇相同的 AWS 區域。
> - AWS Organizations 一次僅支援一個 IAM Identity Center 區域。如果您想要在不同區域中提供 IAM Identity Center，您必須先刪除目前的 IAM Identity Center 組態。如需詳細資訊，請參閱AWS IAM Identity Center 《 使用者指南》中的 IAM Identity Center 區域資料。

中的安全性 AWS IoT Events

的雲端安全性 AWS 是最高優先順序。身為 AWS 客戶，您可以受益於資料中心和網路架構，這些架構專為滿足最安全敏感組織的需求而建置。

安全性是 AWS 與您之間的共同責任。共同責任模型 將此描述為雲端的安全和雲端內的安全：

- 雲端的安全性 – AWS 負責保護在 AWS Cloud 中執行 AWS 服務的基礎設施。 AWS 也為您提供可安全使用的服務。第三方稽核人員定期檢測及驗證安全的效率也是我們 AWS 合規計劃的一部分。若要了解適用的合規計劃 AWS IoT Events，請參閱AWS 合規計劃範圍內的服務。
- 雲端的安全性 – 您的責任取決於您使用 AWS 的服務。您也必須對其他因素負責，包括資料的敏感度、您組織的需求和適用的法律及法規。

本文件將協助您了解如何在使用 時套用共同責任模型 AWS IoT Events。下列主題說明如何設定 AWS IoT Events 以符合您的安全與合規目標。您也將了解如何使用其他 AWS 服務，協助您監控和保護 AWS IoT Events 資源。

主題

- 的身分和存取管理 AWS IoT Events
- 監控 AWS IoT Events 以維護可靠性、可用性和效能
- 的合規驗證 AWS IoT Events
- 中的彈性 AWS IoT Events
- 中的基礎設施安全性 AWS IoT Events

的身分和存取管理 AWS IoT Events

AWS Identity and Access Management (IAM) 是一種 AWS 服務，可協助管理員安全地控制對 AWS 資源的存取。IAM 管理員可控制誰可以進行驗證（登入）和授權（具有許可）來使用 AWS IoT Events 資源。IAM 是一項服務 AWS ，您可以免費使用。

主題

- 目標對象
- 使用身分驗證
- 使用政策管理存取權
- 有關身分和存取管理的更多資訊

- AWS IoT Events 如何使用 IAM
- AWS IoT Events 身分型政策範例
- 的跨服務混淆代理人預防 AWS IoT Events
- 對 AWS IoT Events 身分和存取進行故障診斷

目標對象

使用方式 AWS Identity and Access Management (IAM) 會根據您的角色而有所不同：

- 服務使用者 — 若無法存取某些功能，請向管理員申請所需許可 (請參閱 對 AWS IoT Events 身分和存取進行故障診斷)
- 服務管理員 — 負責設定使用者存取權並提交相關許可請求 (請參閱 AWS IoT Events 如何使用 IAM)
- IAM 管理員 — 撰寫政策以管理存取控制 (請參閱 AWS IoT Events 身分型政策範例)

使用身分驗證

身分驗證是您 AWS 使用身分憑證登入 的方式。您必須以 AWS 帳戶根使用者、IAM 使用者或擔任 IAM 角色身分進行身分驗證。

您可以使用身分來源的登入資料，例如 AWS IAM Identity Center (IAM Identity Center)、單一登入身分驗證或 Google/Facebook 登入資料，以聯合身分的形式登入。如需有關登入的詳細資訊，請參閱《AWS 登入 使用者指南》中的如何登入您的 AWS 帳戶。

對於程式設計存取， AWS 提供 SDK 和 CLI 以密碼編譯方式簽署請求。如需詳細資訊，請參閱《IAM 使用者指南》中的 API 請求的AWS 第 4 版簽署程序。

AWS 帳戶 根使用者

當您建立 時 AWS 帳戶，您會從一個名為 AWS 帳戶 theroot 使用者的登入身分開始，該身分具有對所有 AWS 服務 和 資源的完整存取權。強烈建議不要使用根使用者來執行日常任務。有關需要根使用者憑證的任務，請參閱《IAM 使用者指南》中的需要根使用者憑證的任務。

IAM 使用者和群組

IAM 使用者https://docs.aws.amazon.com/IAM/latest/UserGuide/id_users.html是一種身分具備單人或應用程式的特定許可權。建議以臨時憑證取代具備長期憑證的 IAM 使用者。如需詳細資訊，請參閱《IAM 使用者指南》中的要求人類使用者使用聯合身分提供者來 AWS 使用臨時憑證存取 。

IAM 群組會指定 IAM 使用者集合，使管理大量使用者的許可權更加輕鬆。如需詳細資訊，請參閱《IAM 使用者指南》中的 IAM 使用者的使用案例。

IAM 角色

IAM 角色https://docs.aws.amazon.com/IAM/latest/UserGuide/id_roles.html的身分具有特定許可權，其可以提供臨時憑證。您可以透過從使用者切換到 IAM 角色（主控台）或呼叫 AWS CLI 或 AWS API 操作來擔任角色。如需詳細資訊，請參閱《IAM 使用者指南》中的擔任角色的方法。

IAM 角色適用於聯合身分使用者存取、臨時 IAM 使用者許可、跨帳戶存取權與跨服務存取，以及在 Amazon EC2 執行的應用程式。如需詳細資訊，請參閱《IAM 使用者指南》中的 IAM 中的快帳戶資源存取。

使用政策管理存取權

您可以透過建立政策並將其連接到身分或資源 AWS 來控制 AWS 中的存取。政策定義與身分或資源相關聯的許可。當委託人提出請求時 AWS ， 會評估這些政策。大多數政策會以 JSON 文件 AWS 形式存放在 中。如需進一步了解 JSON 政策文件，請參閱《IAM 使用者指南》中的 JSON 政策概觀。

管理員會使用政策，透過定義哪些主體可在哪些條件下對哪些資源執行動作，以指定可存取的範圍。

預設情況下，使用者和角色沒有許可。IAM 管理員會建立 IAM 政策並將其新增至角色，供使用者後續擔任。IAM 政策定義動作的許可，無論採用何種方式執行。

身分型政策

身分型政策是附加至身分 (使用者、使用者群組或角色) 的 JSON 許可政策文件。這類政策控制身分可對哪些資源執行哪些動作，以及適用的條件。如需了解如何建立身分型政策，請參閱《IAM 使用者指南》中的透過客戶管理政策定義自訂 IAM 許可。

身分型政策可分為內嵌政策 (直接內嵌於單一身分) 與受管政策 (可附加至多個身分的獨立政策)。如需了解如何在受管政策及內嵌政策之間做選擇，請參閱《IAM 使用者指南》中的在受管政策與內嵌政策之間選擇。

其他政策類型

AWS 支援其他政策類型，可設定更多常見政策類型授予的最大許可：

- 許可界限 — 設定身分型政策可授與 IAM 實體的最大許可。如需詳細資訊，請參閱《 IAM 使用者指南》中的 IAM 實體許可界限。

- 服務控制政策 (SCP) — 為 AWS Organizations中的組織或組織單位指定最大許可。如需詳細資訊，請參閱《AWS Organizations 使用者指南》中的服務控制政策。
- 資源控制政策 (RCP) — 設定您帳戶中資源可用許可的上限。如需詳細資訊，請參閱《AWS Organizations 使用者指南》中的資源控制政策 (RCP)。
- 工作階段政策 — 在以程式設計方式為角色或聯合身分使用者建立臨時工作階段時，以參數形式傳遞的進階政策。如需詳細資訊，請參《IAM 使用者指南》中的工作階段政策。

多種政策類型

當多種類型的政策適用於請求時，產生的許可會更複雜而無法理解。若要了解如何 AWS 決定是否在涉及多個政策類型時允許請求，請參閱《IAM 使用者指南》中的政策評估邏輯。

有關身分和存取管理的更多資訊

如需 的身分和存取管理的詳細資訊 AWS IoT Events，請繼續前往下列頁面：

- AWS IoT Events 如何使用 IAM
- 對 AWS IoT Events 身分和存取進行故障診斷

AWS IoT Events 如何使用 IAM

在您使用 IAM 管理對 的存取之前 AWS IoT Events，您應該了解可使用哪些 IAM 功能 AWS IoT Events。若要全面了解 AWS IoT Events 和其他 AWS 服務如何與 IAM 搭配使用，請參閱《AWS IAM 使用者指南》中的與 IAM 搭配使用的 服務。

主題

- AWS IoT Events 身分型政策
- AWS IoT Events 資源型政策
- 以 AWS IoT Events 標籤為基礎的授權
- AWS IoT Events IAM 角色

AWS IoT Events 身分型政策

使用 IAM 身分類型政策，您可以指定允許或拒絕的動作和資源，以及在何種條件下會允許或拒絕動作。 AWS IoT Events 支援特定動作、資源及條件索引鍵。若要了解您在 JSON 政策中使用的所有元素，請參閱《IAM 使用者指南》中的 IAM JSON 政策元素參考。

動作

IAM 身分類型政策的 Action 元素會描述政策將允許或拒絕的特定動作。政策動作通常具有與相關聯 AWS API 操作相同的名稱。政策會使用動作來授予執行相關聯操作的許可。

中的政策動作在動作之前 AWS IoT Events 使用下列字首：iotevents:。例如，若要授予某人使用 API AWS IoT Events CreateInput操作建立 AWS IoT Events 輸入的許可，請在其政策中包含 iotevents:CreateInput動作。若要授予某人使用 API AWS IoT Events BatchPutMessage操作傳送輸入的許可，請在其政策中包含 iotevents-data:BatchPutMessage動作。政策陳述式必須包含 Action或 NotAction元素。 AWS IoT Events 會定義自己的一組動作，描述您可以使用此服務執行的任務。

若要在單一陳述式中指定多個動作，請用逗號分隔，如下所示：

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

您也可以使用萬用字元 (*) 來指定多個動作。例如，若要指定開頭是 Describe 文字的所有動作，請包含以下動作：

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

若要查看 AWS IoT Events 動作清單，請參閱《IAM 使用者指南》中的 定義的動作 AWS IoT Events。

Resources

Resource 元素可指定動作套用的物件。陳述式必須包含 Resource 或 NotResource 元素。您可以使用 ARN 來指定資源，或是使用萬用字元 (*) 來指定陳述式套用到所有資源。

AWS IoT Events 偵測器模型資源具有下列 ARN：

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

如需 ARNs 格式的詳細資訊，請參閱使用 Amazon Resource Name (ARNs) 識別 AWS 資源。

例如，若要在陳述式中指定Foobar偵測器模型，請使用下列 ARN：

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

若要指定屬於特定帳戶的所有執行個體，請使用萬用字元 (*)：

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

有些 AWS IoT Events 動作無法對特定資源執行，例如用於建立資源的動作。在這些情況下，您必須使用萬用字元 (*)。

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

有些 AWS IoT Events API 動作涉及多個資源。例如， 會在其條件陳述式中`CreateDetectorModel`參考輸入，因此使用者必須具有使用輸入和偵測器模型的許可。若要在單一陳述式中指定多項資源，請使用逗號分隔 ARN。

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

若要查看 AWS IoT Events 資源類型及其 ARNs的清單，請參閱《IAM 使用者指南》中的 定義的資源 AWS IoT Events。若要了解您可以使用哪些動作指定每個資源的 ARN，請參閱 AWS IoT Events定義的動作。

條件索引鍵

Condition 元素 (或 Condition 區塊) 可讓您指定使陳述式生效的條件。Condition 元素是選用項目。您可以建置使用條件運算子的條件表達式 (例如等於或小於)，來比對政策中的條件和請求中的值。

若您在陳述式中指定多個 Condition 元素，或是在單一 Condition 元素中指定多個索引鍵， AWS 會使用邏輯 AND 操作評估他們。如果您為單一條件索引鍵指定多個值， 會使用邏輯OR操作 AWS 評估條件。必須符合所有條件，才會授與陳述式的許可。

您也可以在指定條件時使用預留位置變數。例如，您可以只在使用者使用其使用者名稱標記時，將存取資源的許可授予該使用者。如需詳細資訊，請參閱「IAM 使用者指南」中的 IAM 政策元素：變數和標籤。

AWS IoT Events 不提供任何服務特定的條件金鑰，但支援使用一些全域條件金鑰。若要查看所有 AWS 全域條件索引鍵，請參閱《IAM 使用者指南》中的AWS 全域條件內容索引鍵。」

範例

若要檢視 AWS IoT Events 身分型政策的範例，請參閱 AWS IoT Events 身分型政策範例。

AWS IoT Events 資源型政策

AWS IoT Events 不支援以資源為基礎的政策。」 若要檢視詳細資源類型政策頁面的範例，請參閱 https://docs.aws.amazon.com/lambda/latest/dg/access-control-resource-based.html。

以 AWS IoT Events 標籤為基礎的授權

您可以將標籤連接至 AWS IoT Events 資源，或在請求中將標籤傳遞至其中 AWS IoT Events。如需根據標籤控制存取，請使用 `iotevents:ResourceTag/`*`key-name`*、`aws:RequestTag/`*`key-name`* 或 `aws:TagKeys` 條件索引鍵，在政策的條件元素中，提供標籤資訊。如需標記 AWS IoT Events 資源的詳細資訊，請參閱 標記您的 AWS IoT Events 資源。

若要檢視身分型原則範例，以根據該資源上的標籤來限制存取資源，請參閱根據標籤檢視 AWS IoT Events 輸入。

AWS IoT Events IAM 角色

IAM 角色是 中具有特定許可 AWS 帳戶 的實體。

搭配 使用臨時登入資料 AWS IoT Events

您可以搭配聯合使用暫時憑證、擔任 IAM 角色，或是擔任跨帳戶角色。您可以透過呼叫 AssumeRole 或 GetFederationToken 等 AWS Security Token Service AWS STS API 操作來取得臨時安全登入資料。

AWS IoT Events 不支援使用臨時登入資料。

服務連結角色

服務連結角色可讓 AWS 服務存取其他服務中的資源，以代表您完成 動作。服務連結角色會顯示在您的 IAM 帳戶中，並由該服務所擁有。IAM 管理員可以檢視，但不能編輯服務連結角色的許可。

AWS IoT Events 不支援服務連結角色。

服務角色

此功能可讓服務代表您擔任服務角色。此角色可讓服務存取其他服務中的資源，以代表您完成動作。服務角色會出現在您的 IAM 帳戶中，且由該帳戶所擁有。這表示 IAM 管理員可以變更此角色的許可。不過，這樣可能會破壞此服務的功能。

AWS IoT Events 支援服務角色。

AWS IoT Events 身分型政策範例

根據預設，使用者和角色沒有建立或修改 AWS IoT Events 資源的許可。他們也無法使用 AWS 管理主控台 AWS CLI或 AWS API 執行任務。IAM 管理員必須建立 IAM 政策，授予使用者和角色在指定資源上執行特定 API 操作的所需許可。管理員接著必須將這些政策連接至需要這些許可的使用者或群組。

若要了解如何使用這些範例 JSON 政策文件建立 IAM 身分型政策，請參閱《IAM 使用者指南》中的在 JSON 標籤上建立政策。

主題

政策最佳實務

基於身分的政策相當強大。他們會判斷您帳戶中的某個人員是否可以建立、存取或刪除 AWS IoT Events 資源。這些動作可能會讓您的 AWS 帳戶產生費用。當您建立或編輯身分型政策時，請遵循下列準則及建議事項：

- 開始使用 AWS 受管政策 – 若要 AWS IoT Events 快速開始使用 ，請使用 AWS 受管政策為您的員工提供所需的許可。這些政策已在您的帳戶中提供，並由 AWS維護和更新。如需詳細資訊，請參閱《IAM 使用者指南》中的使用許可搭配 AWS 受管政策。
- 授予最低權限 – 當您建立自訂政策時，請只授予執行任務所需要的許可。以最小一組許可開始，然後依需要授予額外的許可。這比一開始使用太寬鬆的許可，稍後再嘗試將他們限縮更為安全。如需詳細資訊，請參閱《IAM 使用者指南》中的授予最低權限。
- 為敏感操作啟用 MFA – 為了提高安全性， 要求使用者使用多重驗證 (MFA) 來存取敏感資源或 API 操作。如需詳細資訊，請參閱《IAM 使用者指南》中的在 AWS中使用多重要素驗證 (MFA)。
- 使用政策條件以增加安全 – 在切實可行的範圍中，請定義您身分類型政策允許存取資源的條件。例如，您可以撰寫條件，指定請求必須來自一定的允許 IP 地址範圍。您也可以撰寫條件，只在指定的日期或時間範圍內允許請求，或是要求使用 SSL 或 MFA。如需詳細資訊，請參閱《IAM 使用者指南》中的 IAM JSON 政策元素：條件。

使用 AWS IoT Events 主控台

若要存取 AWS IoT Events 主控台，您必須擁有一組最低許可。這些許可必須允許您列出和檢視 中 AWS IoT Events 資源的詳細資訊 AWS 帳戶。如果您建立比最基本必要許可更嚴格的身分型政策，則對於具有該政策的實體 (使用者或角色) 而言，主控台就無法如預期運作。

為了確保這些實體仍然可以使用 AWS IoT Events 主控台，請將下列 AWS 受管政策連接至實體。如需詳細資訊，請參閱《IAM 使用者指南》中的新增許可給使用者：

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"
```

```
        }
    ]
}
```

對於僅呼叫 AWS CLI 或 AWS API 的使用者，您不需要允許最低主控台許可。反之，只需允許存取符合您嘗試執行之 API 作業的動作就可以了。

允許使用者在 中檢視自己的許可 AWS IoT Events

此範例會示範如何建立政策，允許 使用者檢視連接到他們使用者身分的內嵌及受管政策。允許使用者檢視自己的 IAM 許可對於安全意識和自助服務功能很有用。此政策包含在主控台或使用 或 AWS CLI AWS API 以程式設計方式完成此動作的許可。

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": "*"
        }
    ]
}
```

存取一個 AWS IoT Events 輸入

對 AWS IoT Events 輸入的精細存取控制對於在多使用者或多團隊環境中維護安全性至關重要。本節說明如何建立 IAM 政策，授予特定 AWS IoT Events 輸入的存取權，同時限制其他人的存取權。

在此範例中，您可以授予使用者 AWS 帳戶 存取其中一個 AWS IoT Events 輸入 `exampleInput`。您也可以允許使用者新增、更新和刪除輸入。

此政策會將 `iotevents:ListInputs`、`iotevents:DescribeInput`、`iotevents:CreateInput`、`iotevents:D` 和 `iotevents:UpdateInput` 許可授予使用者。如需將許可授予使用者並使用主控台進行測試的 Amazon Simple Storage Service (Amazon S3) 範例逐步解說，請參閱使用[使用者政策控制對儲存貯體的存取](#)。

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/*"
        }
    ]
}
```

根據標籤檢視 AWS IoT Events 輸入

標籤可協助您組織 AWS IoT Events 資源。您可以在身分型政策中使用條件，根據標籤控制對 AWS IoT Events 資源的存取。此範例示範如何建立允許檢視##的政策。不過，只有在##標籤Owner的值為該使用者的使用者名稱時，才會授予許可。此政策也會授予在主控台完成此動作的必要許可。

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}"}
            }
        }
    ]
}
```

您可以將此政策連接到您帳戶中的 使用者。如果名為 的使用者`richard-roe`嘗試檢視 AWS IoT Events ##，則##必須加上標籤 `Owner=richard-roe`或 `owner=richard-roe`。否則他便會被拒絕存取。條件標籤鍵 `Owner` 符合 `Owner` 和 `owner`，因為條件索引鍵名稱不區分大小寫。如需詳細資訊，請參閱《IAM 使用者指南》中的 IAM JSON 政策元素：條件。

的跨服務混淆代理人預防 AWS IoT Events

> **Note**
>
> - AWS IoT Events 此服務只允許您使用角色在建立資源的相同帳戶中啟動動作。這有助於防止混淆代理人攻擊 AWS IoT Events。
> - 此頁面可做為參考，讓您了解混淆代理人問題的運作方式，並在服務中 AWS IoT Events 允許跨帳戶資源時加以防止。

混淆代理人問題屬於安全性議題，其中沒有執行動作許可的實體可以強制具有更多許可的實體執行該動作。在 中 AWS，跨服務模擬可能會導致混淆代理人問題。

在某個服務 (呼叫服務) 呼叫另一個服務 (被呼叫服務) 時，可能會發生跨服務模擬。可以操縱呼叫服務來使用其許可，以其不應有存取許可的方式對其他客戶的資源採取動作。為了防止這種情況， AWS 提供工具，協助您保護所有 服務的資料，讓 服務主體能夠存取您帳戶中的資源。

我們建議在資源政策中使用 `aws:SourceArn`和 `aws:SourceAccount`全域條件內容索引鍵，以限制將另一個 服務 AWS IoT Events 提供給資源的許可。如果 `aws:SourceArn` 值不包含帳戶 ID (例如 Amazon S3 儲存貯體 ARN)，您必須使用這兩個全域條件內容金鑰來限制許可。如果同時使用這兩個全域條件內容金鑰，且 `aws:SourceArn` 值包含帳戶 ID，則在相同政策陳述式中使用 `aws:SourceAccount` 值和 `aws:SourceArn` 值中的帳戶時，必須使用相同的帳戶 ID。

如果您想要僅允許一個資源與跨服務存取相關聯，則請使用 `aws:SourceArn`。如果您想要允許該帳戶中的任何資源與跨服務使用相關聯，請使用 `aws:SourceAccount`。的值`aws:SourceArn`必須是與`sts:AssumeRole`請求相關聯的偵測器模型或警示模型。

防範混淆代理人問題最有效的方法，是使用 aws:SourceArn 全域條件內容金鑰，以及資源的完整 ARN。如果不知道資源的完整 ARN，或者如果您指定了多個資源，請使用 aws:SourceArn 全域條件內容金鑰，同時使用萬用字元 (*) 表示 ARN 的未知部分。例如 arn:aws:*iotevents*:*:*123456789012*:*。

下列範例示範如何在 中使用 aws:SourceArn和 aws:SourceAccount全域條件內容索引鍵 AWS IoT Events ，以防止混淆代理人問題。

主題

- 範例：安全存取 AWS IoT Events 偵測器模型
- 範例：安全存取 AWS IoT Events 警示模型
- 範例：存取指定區域中 AWS IoT Events 的資源
- 範例：設定 的記錄選項 AWS IoT Events

範例：安全存取 AWS IoT Events 偵測器模型

此範例示範如何建立 IAM 政策，以安全地將存取權授予 中的特定偵測器模型 AWS IoT Events。此政策使用條件來確保只有指定的 AWS 帳戶 AWS IoT Events 和服務可以擔任該角色，並新增額外的安全層。在此範例中，角色只能存取名為 *WindTurbine01* 的偵測器模型。

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"
                }
            }
        }
    ]
}
```

範例：安全存取 AWS IoT Events 警示模型

此範例示範如何建立 IAM 政策， AWS IoT Events 允許 安全地存取警示模型。此政策使用條件來確保只有指定的 AWS 帳戶 AWS IoT Events 和服務可以擔任該角色。

在此範例中，角色可以存取指定 AWS 帳戶中的任何警示模型，如警示模型 ARN 中的*萬用字元所示。`aws:SourceAccount` 和 `aws:SourceArn`條件共同運作，以防止混淆代理人問題。

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/*"
                }
            }
        }
    ]
}
```

範例：存取指定區域中 AWS IoT Events 的資源

此範例示範如何設定 IAM 角色以存取特定 AWS 區域中 AWS IoT Events 的資源。透過在 IAM 政策中使用區域特定的 ARNs，您可以限制存取不同地理區域 AWS IoT Events 的資源。此方法有助於在多區域部署中維持安全與合規。此範例中的區域是 *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:*"
                }
            }
        }
    ]
}
```

範例：設定 的記錄選項 AWS IoT Events

適當的記錄對於監控、偵錯和稽核您的 AWS IoT Events 應用程式至關重要。本節提供 中可用記錄選項的概觀 AWS IoT Events。

此範例示範如何設定允許 AWS IoT Events 將資料記錄到 CloudWatch Logs 的 IAM 角色。在資源 ARN 中使用萬用字元 (*) 可讓您跨 AWS IoT Events 基礎設施進行全面記錄。

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:*"
                }
            }
        }
    ]
}
```

對 AWS IoT Events 身分和存取進行故障診斷

使用以下資訊來協助您診斷和修正使用 AWS IoT Events 和 IAM 時可能遇到的常見問題。

主題

- 我無權在 中執行動作 AWS IoT Events
- 我未獲得執行 iam:PassRole 的授權
- 我想要允許 以外的人員 AWS 帳戶 存取我的 AWS IoT Events 資源

我無權在 中執行動作 AWS IoT Events

如果 AWS 管理主控台 告知您無權執行 動作，則必須聯絡您的管理員尋求協助。您的管理員是為您提供使用者名稱和密碼的人員。

當 IAM mateojackson 使用者嘗試使用主控台檢視*##*的詳細資訊，但沒有iotevents:*ListInputs*許可時，會發生下列範例錯誤。

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

在此情況下，Mateo 會請求管理員更新他的政策，允許他使用 *my-example-input* 動作存取 iotevents:*ListInput* 資源。

我未獲得執行 **iam:PassRole** 的授權

如果您收到錯誤，告知您未獲授權執行 iam:PassRole 動作，您的政策必須更新，允許您將角色傳遞給 AWS IoT Events。

有些 AWS 服務 可讓您將現有角色傳遞給該服務，而不是建立新的服務角色或服務連結角色。如需執行此作業，您必須擁有將角色傳遞至該服務的許可。

名為 marymajor 的 IAM 使用者嘗試使用主控台在 AWS IoT Events中執行動作時，發生下列範例錯誤。但是，動作要求服務具備服務角色授予的許可。Mary 沒有將角色傳遞給服務的許可。

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

在這種情況下，Mary 的政策必須更新，允許她執行 iam:PassRole 動作。

如果您需要協助，請聯絡您的 AWS 管理員。您的管理員提供您的簽署憑證。

我想要允許 以外的人員 AWS 帳戶 存取我的 AWS IoT Events 資源

您可以建立一個角色，讓其他帳戶中的使用者或您組織外部的人員存取您的資源。您可以指定要允許哪些信任物件取得該角色。針對支援基於資源的政策或存取控制清單 (ACL) 的服務，您可以使用那些政策來授予人員存取您的資源的許可。

請參閱下列主題以判斷您的最佳選項：

- 若要了解 是否 AWS IoT Events 支援這些功能，請參閱 AWS IoT Events 如何使用 IAM。
- 若要了解如何提供您擁有 AWS 帳戶 的資源存取權，請參閱《IAM 使用者指南》中的在您擁有 AWS 帳戶 的另一個 中為 IAM 使用者提供存取權。
- 若要了解如何將資源的存取權提供給第三方 AWS 帳戶，請參閱《IAM 使用者指南》中的將存取權提供給第三方 AWS 帳戶 擁有。

- 如需了解如何透過聯合身分提供存取權，請參閱《IAM 使用者指南》中的將存取權提供給在外部進行身分驗證的使用者 (聯合身分)。
- 如需了解使用角色和資源型政策進行跨帳戶存取之間的差異，請參閱《IAM 使用者指南》中的 IAM 中的跨帳戶資源存取。

監控 AWS IoT Events 以維護可靠性、可用性和效能

監控是維護 和 AWS 解決方案的可靠性、可用性 AWS IoT Events 和效能的重要部分。您應該從 AWS 解決方案的所有部分收集監控資料，以便在發生多點失敗時更輕鬆地偵錯。開始監控之前 AWS IoT Events，您應該建立監控計畫，其中包含下列問題的答案：

- 監控目標是什麼？
- 監控哪些資源？
- 監控這些資源的頻率為何？
- 將使用哪些監控工具？
- 誰將執行監控任務？
- 發生問題時應該通知誰？

下一個步驟是為環境中的正常 AWS IoT Events 效能建立基準，方法是在不同的時間和不同的負載條件下測量效能。當您監控 AWS IoT Events時，請存放歷史記錄監控資料，如此才能與目前的效能資料做比較、辨識正常效能模式和效能異常狀況、規劃問題處理方式。

例如，如果您使用的是 Amazon EC2，您可以監控執行個體的 CPU 使用率、磁碟 I/O 和網路使用率。若效能不符合您所建立的基準，您可能需要重新設定或將執行個體最佳化，以降低 CPU 使用率、改善磁碟 I/O、降低網路流量。

主題

- 可監控的可用工具 AWS IoT Events
- AWS IoT Events 使用 Amazon CloudWatch 進行監控
- 使用 記錄 AWS IoT Events API 呼叫 AWS CloudTrail

可監控的可用工具 AWS IoT Events

AWS 提供各種可用來監控的工具 AWS IoT Events。您可以設定其中一些工具來進行監控，但有些工具需要手動介入。建議您盡可能自動化監控任務。

自動化監控工具

您可以使用下列自動化監控工具，在發生錯誤時監看 AWS IoT Events 和報告 ：

- Amazon CloudWatch Logs – 從 AWS CloudTrail 或其他來源監控、存放和存取您的日誌檔案。如需詳細資訊，請參閱《Amazon CloudWatch 使用者指南》中的使用 Amazon CloudWatch 儀表板。
- AWS CloudTrail 日誌監控 – 在帳戶之間共用日誌檔案、透過將日誌檔案傳送到 CloudWatch Logs 來即時監控 CloudTrail 日誌檔案、在 Java 中寫入日誌處理應用程式，以及驗證您的日誌檔案在 CloudTrail 交付後並未變更。如需詳細資訊，請參閱 AWS CloudTrail 使用者指南中的使用 CloudTrail 日誌檔案。

手動監控工具

監控的另一個重要部分 AWS IoT Events 包括手動監控 CloudWatch 警示未涵蓋的項目。 AWS IoT Events、CloudWatch 和其他 AWS 主控台儀表板提供 AWS 環境狀態的at-a-glance。建議您也檢查 上的日誌檔案 AWS IoT Events。

- AWS IoT Events 主控台會顯示：
  - 偵測器模型
  - 偵測器
  - 輸入
  - 設定
- CloudWatch 首頁會顯示：
  - 目前警示與狀態
  - 警示與資源的圖表
  - 服務運作狀態

  此外，您可以使用 CloudWatch 執行下列動作：
  - 建立建立 CloudWatch 儀表板以監控您關心的服務
  - 用於疑難排解問題以及探索驅勢的圖形指標資料。
  - 搜尋和瀏覽您的所有 AWS 資源指標
  - 建立與編輯要通知發生問題的警示

AWS IoT Events 使用 Amazon CloudWatch 進行監控

當您開發或偵錯 AWS IoT Events 偵測器模型時，您需要知道 AWS IoT Events 正在做什麼，以及它遇到的任何錯誤。Amazon CloudWatch AWS 會即時監控您的 AWS 資源和您在 上執行的應用程式。透過 CloudWatch，您可以全面了解資源使用、應用程式效能和運作狀態。 在開發 AWS IoT Events 偵測器模型時啟用 Amazon CloudWatch 記錄 提供如何為 啟用 CloudWatch 記錄的相關資訊 AWS IoT Events。若要產生類似如下所示的日誌，您必須將詳細程度設定為「偵錯」，並提供一或多個偵錯目標，即偵測器模型名稱和選用的 KeyValue。

下列範例顯示 產生的 CloudWatch DEBUG 層級日誌項目 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 記錄 AWS IoT Events API 呼叫 AWS CloudTrail

AWS IoT Events 已與 服務整合 AWS CloudTrail，此服務提供使用者、角色或 AWS 服務在其中採取之動作的記錄 AWS IoT Events。CloudTrail 會將 的所有 API 呼叫擷取 AWS IoT Events 為事件，包括來自 AWS IoT Events 主控台的呼叫，以及來自對 AWS IoT Events APIs的程式碼呼叫。

如果您建立線索，您可以將 CloudTrail 事件持續交付至 Amazon S3 儲存貯體，包括 的事件 AWS IoT Events。即使您未設定追蹤，依然可以透過 CloudTrail 主控台中的事件歷史記錄檢視最新事件。您可以使用 CloudTrail 所收集的資訊來判斷提出的請求 AWS IoT Events、提出請求的 IP 地址、提出請求的人員、提出請求的時間，以及其他詳細資訊。

若要進一步了解 CloudTrail，請參閱AWS CloudTrail 《使用者指南》。

AWS IoT Events CloudTrail 中的資訊

當您建立 AWS 帳戶時，會在您的帳戶上啟用 CloudTrail。當活動在 中發生時 AWS IoT Events，該活動會記錄在 CloudTrail 事件中，並在事件歷史記錄中記錄其他 AWS 服務事件。您可以在 AWS 帳戶中檢視、搜尋和下載最近的事件。如需詳細資訊，請參閱使用 CloudTrail 事件歷史記錄。

若要持續記錄您 AWS 帳戶中的事件，包括 的事件 AWS IoT Events，請建立追蹤。線索能讓 CloudTrail 將日誌檔案交付至 Amazon S3 儲存貯體。根據預設，當您在主控台中建立線索時，線索會套用至所有 AWS 區域。線索會記錄 AWS 分割區中所有區域的事件，並將日誌檔案傳送到您指定的 Amazon S3 儲存貯體。此外，您可以設定其他 AWS 服務，以進一步分析和處理 CloudTrail 日誌中所收集的事件資料。如需詳細資訊，請參閱：

- 為 AWS 您的帳戶建立追蹤
- CloudTrail 支援的服務和整合
- 設定 CloudTrail 的 Amazon SNS 通知
- 接收多個區域的 CloudTrail 日誌檔案和接收多個帳戶的 CloudTrail 日誌檔案

每一筆事件或日誌專案都會包含產生請求者的資訊。身分資訊可協助您判斷下列事項：

- 該請求是否使用根或 IAM 使用者憑證提出。
- 提出該請求時，是否使用了特定角色或聯合身分使用者的暫時安全憑證。
- 請求是否由其他 AWS 服務提出。

如需詳細資訊，請參閱 CloudTrail userIdentity element. AWS IoT Events actions 記錄在 AWS IoT Events API 參考中。

了解 AWS IoT Events 日誌檔案項目

追蹤是一種組態，可讓您將事件做為日誌檔案交付至您指定的 Amazon S3 儲存貯體。 AWS CloudTrail 日誌檔案包含一或多個日誌項目。一個事件為任何來源提出的單一請求，並包含請求動作、請求的日期和時間、請求參數等資訊。CloudTrail 日誌檔案並非依公有 API 呼叫追蹤記錄的堆疊排序，因此不會以任何特定順序出現。

當 AWS 您的帳戶中啟用 CloudTrail 記錄時，對 AWS IoT Events 動作進行的大多數 API 呼叫都會在 CloudTrail 日誌檔案中追蹤，而這些日誌檔案會與其他 AWS 服務記錄一起寫入。CloudTrail 會根據期間與檔案大小，決定何時建立與寫入新檔案。

每個日誌項目都會包含產生要求之人員的資訊。日誌記錄中的使用者身分資訊，可協助您判斷下列事項：

- 該請求是否使用根或 IAM 使用者憑證提出。
- 提出該請求時，是否使用了特定角色或聯合身分使用者的暫時安全憑證。
- 請求是否由其他 AWS 服務提出。

您可以視需要將日誌檔案存放在 Amazon S3 儲存貯體中，但您也可以定義 Amazon S3 生命週期規則來自動封存或刪除日誌檔案。您的日誌檔案預設使用 Amazon S3 伺服器端加密 (SSE) 加密。

若要在日誌檔案交付時收到通知，您可以設定 CloudTrail 在新日誌檔案交付時發佈 Amazon SNS 通知。如需詳細資訊，請參閱為 CloudTrail 設定 Amazon SNS 通知。

您也可以將多個 AWS 區域和多個 AWS 帳戶的 AWS IoT Events 日誌檔案彙整至單一 Amazon S3 儲存貯體。

詳情請參閱從多個區域接收 CloudTrail 日誌檔案，以及從多個帳戶接收 CloudTrail 日誌檔案。

範例：CloudTrail 的 DescribeDetector 動作

以下範例顯示的是展示 `DescribeDetector` 動作的 CloudTrail 日誌項目。


```
{
  "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"
}
```

範例：CloudTrail 的 CreateDetectorModel 動作

以下範例顯示的是展示 `CreateDetectorModel` 動作的 CloudTrail 日誌項目。

```
{
  "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"
}
```

範例：CloudTrail 的 CreateInput 動作

以下範例顯示的是展示 `CreateInput` 動作的 CloudTrail 日誌項目。

```
{
  "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"
}
```

範例：CloudTrail 的 DeleteDetectorModel 動作

以下範例顯示的是展示 `DeleteDetectorModel` 動作的 CloudTrail 日誌項目。

```
{
  "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"
}
```


範例：CloudTrail 的 DeleteInput 動作

以下範例顯示的是展示 `DeleteInput` 動作的 CloudTrail 日誌項目。


```
{
  "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"
}
```

範例：CloudTrail 的 DescribeDetectorModel 動作

以下範例顯示的是展示 `DescribeDetectorModel` 動作的 CloudTrail 日誌項目。

```
{
  "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"
}
```

範例：CloudTrail 的 DescribeInput 動作

以下範例顯示的是展示 `DescribeInput` 動作的 CloudTrail 日誌項目。

```
{
  "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"
}
```


範例：CloudTrail 的 DescribeLoggingOptions 動作

以下範例顯示的是展示 `DescribeLoggingOptions` 動作的 CloudTrail 日誌項目。


```
{
  "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"
}
```

範例：CloudTrail 的 ListDetectorModels 動作

以下範例顯示的是展示 `ListDetectorModels` 動作的 CloudTrail 日誌項目。

```
{
  "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"
}
```

範例：CloudTrail 的 ListDetectorModelVersions 動作

以下範例顯示的是展示 `ListDetectorModelVersions` 動作的 CloudTrail 日誌項目。

```
{
  "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"
}
```

範例：CloudTrail 的 ListDetectors 動作

以下範例顯示的是展示 `ListDetectors` 動作的 CloudTrail 日誌項目。

```
{
  "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"
}
```

範例：CloudTrail 的 ListInputs 動作

以下範例顯示的是展示 `ListInputs` 動作的 CloudTrail 日誌項目。

```
{
  "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"
}
```

範例：CloudTrail 的 PutLoggingOptions 動作

以下範例顯示的是展示 PutLoggingOptions 動作的 CloudTrail 日誌項目。

```
{
  "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"
}
```

範例：CloudTrail 的 UpdateDetectorModel 動作

以下範例顯示的是展示 `UpdateDetectorModel` 動作的 CloudTrail 日誌項目。

```
{
  "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"
}
```

範例：CloudTrail 的 UpdateInput 動作

以下範例顯示的是展示 `UpdateInput` 動作的 CloudTrail 日誌項目。

```
{
  "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"
}
```

範例：CloudTrail 的 BatchPutMessage 動作

AWS IoT Events 可以使用 CloudTrail 整合進行資料平面 API 記錄。此範例會透過 `BatchPutMessage`動作新增資料事件的詳細資訊。

```
{
            "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"
        }
```

```
        },
```

的合規驗證 AWS IoT Events

若要了解 是否 AWS 服務 在特定合規計劃的範圍內，請參閱AWS 服務 合規計劃範圍內然後選擇您感興趣的合規計劃。如需一般資訊，請參閱AWS 合規計劃。

您可以使用 下載第三方稽核報告 AWS Artifact。如需詳細資訊，請參閱在 中下載報告 AWS Artifact。

您使用 時的合規責任 AWS 服務 取決於資料的機密性、您公司的合規目標，以及適用的法律和法規。如需使用 時合規責任的詳細資訊 AWS 服務，請參閱 AWS 安全文件。

中的彈性 AWS IoT Events

AWS 全域基礎設施是以 AWS 區域和可用區域為基礎。 AWS 區域提供多個分開且隔離的實際可用區域，它們以低延遲、高輸送量和高度備援聯網功能相互連結。透過可用區域，您所設計與操作的應用程式和資料庫，就能夠在可用區域之間自動容錯移轉，而不會發生中斷。可用區域的可用性、容錯能力和擴充能力，均較單一或多個資料中心的傳統基礎設施還高。

如需 AWS 區域和可用區域的詳細資訊，請參閱AWS 全域基礎設施。

中的基礎設施安全性 AWS IoT Events

作為受管服務， AWS IoT Events 受到 AWS 全球網路安全的保護。如需 AWS 安全服務以及如何 AWS 保護基礎設施的資訊，請參閱AWS 雲端安全。若要使用基礎設施安全的最佳實務來設計您的 AWS 環境，請參閱安全支柱 AWS Well-Architected Framework 中的基礎設施保護。

您可以使用 AWS 發佈的 API 呼叫， AWS IoT Events 透過網路存取 。使用者端必須支援下列專案：

- Transport Layer Security (TLS)。我們需要 TLS 1.2 並建議使用 TLS 1.3。
- 具備完美轉送私密(PFS)的密碼套件，例如 DHE (Ephemeral Diffie-Hellman)或 ECDHE (Elliptic Curve Ephemeral Diffie-Hellman)。現代系統(如 Java 7 和更新版本)大多會支援這些模式。

AWS AWS IoT Events 資源的服務配額

AWS 一般參考 指南提供 AWS IoT Events AWS 帳戶的預設配額。除非另有說明，否則每個配額都是每個 AWS 區域。如需詳細資訊，請參閱《 AWS 一般參考 指南》中的AWS IoT Events 端點、配額AWS Service Quotas。

若要請求提高服務配額，請在支援中心主控台中提交支援案例。如需詳細資訊，請參閱「Service Quotas 使用者指南」中的請求提高配額。

> **Note**
>
> - 偵測器模型和輸入的所有名稱在帳戶中必須是唯一的。
> - 您無法在偵測器模型和輸入建立後變更它們的名稱。

標記您的 AWS IoT Events 資源

為了協助您管理和組織偵測器模型和輸入，您可以選擇以標籤形式將自己的中繼資料指派給每個資源。本節說明標籤並示範如何建立它們。

標籤基本概念

標籤可讓您以不同的方式分類 AWS IoT Events 資源，例如依用途、擁有者或環境。這在您擁有許多相同類型的資源時很有用。您可以根據您指派給資源的標籤快速識別特定資源。

每個標籤皆包含由您定義的一個「索引鍵」與選擇性的「值」。例如，您可以為輸入定義一組標籤，協助您依類型追蹤傳送這些輸入的裝置。我們建議您為每種資源類型建立符合您需求的一組標籤金鑰。使用一致的標籤金鑰組可讓您更輕鬆管理您的 資源。

您可以根據您新增或套用的標籤來搜尋和篩選資源、使用標籤來分類和追蹤成本，以及使用標籤來控制對資源的存取，如《 AWS IoT 開發人員指南》中的使用標籤搭配 IAM 政策中所述。

為了方便使用， 中的標籤編輯器 AWS 管理主控台 提供集中、統一的方式來建立和管理標籤。如需詳細資訊，請參閱《標記 AWS 資源和標籤編輯器使用者指南》中的標籤編輯器入門。

您也可以使用 AWS CLI 和 AWS IoT Events API 處理標籤。使用下列命令中的 "Tags" 欄位建立標籤時，您可以將標籤與偵測器模型和輸入建立關聯：

- CreateDetectorModel
- CreateInput

您可以使用下列命令新增、修改或刪除支援標記功能的現有資源標籤：

- TagResource
- ListTagsForResource
- UntagResource

您可以編輯標籤索引鍵和值，也可以隨時從資源中移除標籤。您可以將標籤的值設為空白字串，但您無法將標籤的值設為 Null。如果您新增的標籤具有與該資源上現有標籤相同的索引鍵，則新值會覆寫舊值。如果您刪除資源，也會刪除與該資源相關聯的任何標籤。

如需詳細資訊，請參閱標記 AWS 資源的最佳實務

標籤的限制與上限

以下基本限制適用於 標籤：

- 每一資源最多標籤數：50
- 金鑰長度上限 – UTF-8 127 個 Unicode 字元
- 最大值長度 – 255 個 UTF-8 Unicode 字元
- 標籤鍵與值皆區分大小寫。
- 請勿在標籤名稱或值中使用 "aws:"字首，因為它已保留供 AWS 使用。您不可編輯或刪除具此字首的標籤名稱或值。具此字首的標籤，不算在受資源限制的標籤計數內。
- 如果您的標記結構描述是跨多項服務和資源使用，請記得其他服務可能會有字元使用限制。通常，允許使用的字元為：可用 UTF-8 表示的英文字母、空格和數字，以及以下特殊字元：+ - = . _ : / @。

搭配 IAM 政策使用標籤

您可以在用於 AWS IoT Events API 動作的 IAM 政策中，套用以標籤為基礎的資源層級許可。這可讓您更有效地控制使用者可以建立、修改或使用哪些資源。

您可以使用 Condition 元素 (也稱為 Condition 區塊)，以及 IAM 政策中的以下條件內容金鑰和值，來根據資源標籤控制使用者存取 (許可)：

- 使用 `aws:ResourceTag/<tag-key>: <tag-value>` 以允許或拒絕資源上具有特定標籤的使用者動作。
- 使用 `aws:RequestTag/<tag-key>: <tag-value>` 以在提出 API 請求時，要求使用 (或不使用) 特定標籤，以建立或修改允許標籤的資源。
- 使用 `aws:TagKeys: [<tag-key>, ...]` 以在提出 API 請求時，要求使用 (或不使用) 特定標籤金鑰集，以建立或修改允許標籤的資源。

> **Note**
>
> IAM 政策中的條件內容金鑰和值，只會套用到資源識別符可標記為必要參數的那些 AWS IoT Events 動作。

AWS Identity and Access Management 《使用者指南》中的使用標籤控制存取，有使用標籤的其他資訊。該指南的 IAM JSON 政策參考章節有詳細的語法、說明，還有元素、變數範例，以及在 IAM 中的 JSON 政策評估邏輯。

以下範例政策會套用兩個以標籤為基礎的限制。此政策限制的使用者：

- 無法給予資源 "env = prod" 標籤 (在範例中，請參閱此行 `"aws:RequestTag/env" : "prod"`
- 無法修改或存取包含現有標籤 "env = prod" 的資源 (在範例中，請參閱此行 `"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": "*"
        }
    ]
}
```

您也可以透過將指定標籤索引鍵括在清單中來指定多個標籤值，如下所示。

```
        "StringEquals" : {
          "aws:ResourceTag/env" : ["dev", "test"]
        }
```

> **Note**
>
> 如果您允許或拒絕使用者根據標籤存取資源，請務必考慮明確拒絕使用者將這些標籤新增至相同資源或從中移除的能力。否則，使用者可能透過修改標籤來避開您的限制，並取得資源的存取。

故障診斷 AWS IoT Events

本故障診斷指南提供您在使用 時可能遇到的常見問題的解決方案 AWS IoT Events。瀏覽主題以識別和解決偵測事件、存取資料、許可、服務整合、裝置組態等問題。本指南提供 AWS IoT Events 主控台、API、CLI、錯誤、延遲和整合的故障診斷建議，旨在快速解決問題，以便您可以建置可靠且可擴展的事件驅動型應用程式。

主題

AWS IoT Events 常見問題和解決方案

請參閱下一節，針對錯誤進行故障診斷，並尋找解決 問題的可能解決方案 AWS IoT Events。

錯誤

偵測器模型建立錯誤

當我嘗試建立偵測器模型時出現錯誤。

解決方案

建立偵測器模型時，您必須考慮下列限制。

- 每個action欄位只允許一個動作。
- condition 是 的必要項目transitionEvents。OnEnter、 OnInput和 OnExit事件是選用的。
- 如果 condition 欄位為空，則條件表達式的評估結果等同於 true。
- 條件表達式的評估結果應為布林值。如果結果不是布林值，則等於 false 且 不會觸發 actions或轉換為事件中nextState指定的 。

如需詳細資訊，請參閱AWS IoT Events 偵測器模型限制。

來自已刪除偵測器模型的更新

我幾分鐘前更新或刪除了偵測器模型，但我仍透過 MQTT 訊息或 SNS 提醒從舊偵測器模型取得狀態更新。

解決方案

如果您更新、刪除或重新建立偵測器模型 （ 請參閱 UpdateDetectorModel)，則會在刪除所有偵測器執行個體並使用新模型之前發生延遲。在此期間，舊版偵測器模型的執行個體可能會繼續處理輸入。您可能會繼續收到先前偵測器模型定義的提醒。請等待至少 7 分鐘，再重新檢查更新或報告錯誤。

動作觸發失敗 （ 符合條件時 ）

當條件滿足時，偵測器無法觸發動作或轉換為新狀態。

解決方案

確認偵測器條件式表達式的評估結果是布林值。如果結果不是布林值，則相當於 false，而且 不會觸發 action或 轉換為事件中nextState指定的 。如需詳細資訊，請參閱條件式表達式語法。

動作觸發失敗 （ 達到閾值時 ）

當條件式表達式中的變數達到指定的值時，偵測器不會觸發動作或事件轉換。

解決方案

如果您setVariable更新 onInput、 onEnter或 onExit，則在目前處理週期condition期間評估任何 時，不會使用新的值。而是使用原始值，直到目前的週期完成為止。您可以在偵測器模型定義中

設定 evaluationMethod 參數，以變更此行為。當 evaluationMethod 設定為 時SERIAL，變數會更新，並依事件定義的順序評估事件條件。當 evaluationMethod 設為 BATCH（預設值）時，只有在評估所有事件條件後，才會更新變數並執行事件。

狀態用量不正確

當我嘗試使用 傳送訊息至輸入時，偵測器會進入錯誤狀態BatchPutMessage。

解決方案

如果您使用 BatchPutMessage 將多則訊息傳送至輸入，無法保證訊息或輸入的處理順序。若要保證訂購，請一次傳送訊息一個，每次等待 BatchPutMessage 確認成功。

連線訊息

當我嘗試呼叫或叫用 API 時收到('Connection aborted.', error(54, 'Connection reset by peer'))錯誤。

解決方案

確認 OpenSSL 使用 TLS 1.1 或更新版本來建立連線。這應該是大多數 Linux 發行版本或 Windows 第 7 版及更新版本下的預設值。macOS 使用者可能需要升級 OpenSSL。

InvalidRequestException 訊息

當我嘗試呼叫 CreateDetectorModel和 UpdateDetectorModel APIs時，會收到 InvalidRequestException。

解決方案

檢查下列項目以協助解決問題。如需詳細資訊，請參閱 CreateDetectorModel 和 UpdateDetectorModel。

- 請確定您不同時使用 seconds和 durationExpression做為 的參數SetTimerAction。
- 請確定 的字串表達durationExpression式有效。字串表達式可以包含數字、變數 ($variable.<variable-name>) 或輸入值 ($input.<input-name>.<path-to-datum>)。

Amazon CloudWatch Logs **action.setTimer**錯誤

您可以設定 Amazon CloudWatch Logs 來監控 AWS IoT Events 偵測器模型執行個體。以下是當您使用 時 AWS IoT Events， 產生的常見錯誤action.setTimer。

- 錯誤：您的名為 的計時器持續時間表達式*<timer-name>*無法評估為數字。

解決方案

請確定您的 字串表達式durationExpression可以轉換為數字。不允許其他資料類型，例如布林值。

- 錯誤：名為 的計時器的持續時間表達式評估結果*<timer-name>*大於 31622440。為了確保準確性，請確定您的持續時間表達式參考介於 60-31622400 之間的值。

解決方案

請確定計時器的持續時間小於或等於 31622400 秒。持續時間的評估結果會四捨五入到最接近的整數。

- 錯誤：名為 之計時器的持續時間表達式評估結果*<timer-name>*小於 60。為了確保準確性，請確定您的持續時間表達式參考介於 60-31622400 之間的值。

解決方案

請確定計時器的持續時間大於或等於 60 秒。持續時間的評估結果會四捨五入到最接近的整數。

- 錯誤：*<timer-name>*無法評估名為 之計時器的持續時間表達式。檢查資料的變數名稱、輸入名稱和路徑，以確保您參考現有的變數和輸入。

解決方案

請確定您的字串表達式參考現有的變數和輸入。字串表達式可以包含數字、變數 ($variable.*variable-name*) 和輸入值 ($input.*input-name*.*path-to-datum*)。

- 錯誤：無法設定名為 的計時器<timer-name>。請檢查您的持續時間表達式，然後再試一次。

解決方案

請參閱 SetTimerAction 動作以確保您指定了正確的參數，然後再次設定計時器。

如需詳細資訊，請參閱在開發 AWS IoT Events 偵測器模型時啟用 Amazon CloudWatch 記錄。

Amazon CloudWatch 承載錯誤

您可以設定 Amazon CloudWatch Logs 來監控 AWS IoT Events 偵測器模型執行個體。以下是當您設定動作承載時 AWS IoT Events， 產生的常見錯誤和警告。

- 錯誤：我們無法評估動作的表達式。確定資料的變數名稱、輸入名稱和路徑參考現有的變數和輸入值。此外，請確認承載的大小小於 1 KB，即承載允許的大小上限。

解決方案

請確定您輸入正確的變數名稱、輸入名稱和資料路徑。如果動作承載大於 1 KB，您也可能收到此錯誤訊息。

- 錯誤：我們無法剖析 承載的內容表達式*<action-type>*。使用正確的語法輸入內容表達式。

解決方案

內容表達式可以包含字串 ('*string*')、變數 ($variable.*variable-name*)、輸入值 ($input.*input-name*.*path-to-datum*)、字串串連，以及包含 的字串${}。

- 錯誤：您的承載表達式 {*expression*} 無效。定義的承載類型為 JSON，因此您必須指定 AWS IoT Events 評估為字串的表達式。

解決方案

如果指定的承載類型是 JSON，則 AWS IoT Events 首先檢查服務是否可以評估字串的表達式。評估的結果不能是布林值或數字。如果驗證失敗，您可能會收到此錯誤。

- 警告：動作已執行，但我們無法將動作承載的內容表達式評估為有效的 JSON。定義的承載類型為 JSON。

解決方案

如果您將承載類型定義為 ，請確定 AWS IoT Events 可以將動作承載的內容表達式評估為有效的 JSONJS0N。即使 AWS IoT Events 無法將內容表達式評估為有效的 JSON， 仍會 AWS IoT Events 執行動作。

如需詳細資訊，請參閱在開發 AWS IoT Events 偵測器模型時啟用 Amazon CloudWatch 記錄。

不相容的資料類型

訊息：在下列表達式<reference>中找到 的不相容資料類型 【<inferred-types>】：<expression>

解決方案

您可能會因為下列其中一個原因收到此錯誤：

- 參考的評估結果與表達式中的其他運算元不相容。
- 不支援傳遞至函數的引數類型。

當您在表達式中使用參考時，請檢查下列項目：

- 當您使用參考做為具有一或多個運算子的運算元時，請確定您參考的所有資料類型都相容。

  例如，在以下表達式中，整數2是 ==和 &&運算子的運算元。為了確保運算元相容，`$variable.testVariable + 1`且`$variable.testVariable`必須參考整數或小數。

  此外，整數1是+運算子的運算元。因此，`$variable.testVariable` 必須參考整數或小數。

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

- 當您使用參考做為傳遞給函數的引數時，請確定該函數支援您參考的資料類型。

  例如，下列`timeout("time-name")`函數需要具有雙引號的字串做為引數。如果您使用`#####`值的參考，則必須參考具有雙引號的字串。

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

  > **Note**
  >
  > 對於 `convert(type, expression)`函數，如果您使用`##`值的參考，則參考的評估結果必須是 `String`、 `Decimal`或 `Boolean`。

如需詳細資訊，請參閱AWS IoT Events 運算式中輸入和變數的參考。

無法傳送訊息至 AWS IoT Events

訊息：無法傳送訊息至 Iot Events

解決方案

您可能會因為下列原因遇到此錯誤：

- 輸入訊息承載不包含 `Input attribute Key`。
- `Input attribute Key` 與輸入定義中指定的 JSON 路徑不同。

- 輸入訊息與 AWS IoT Events 輸入中定義的結構描述不相符。

> **Note**
>
> 從其他 服務擷取的資料也會失敗。

Example

例如，在 中 AWS IoT Core， AWS IoT 規則會失敗並顯示下列訊息 `Verify the Input Attribute key.`

若要解決此問題，請確定輸入承載訊息結構描述符合 AWS IoT Events 輸入定義，且`Input attribute Key`位置相符。如需詳細資訊，請參閱 以在 中建立模型的輸入 AWS IoT Events了解如何定義 AWS IoT Events 輸入。

透過在 中執行分析對偵測器模型進行故障診斷 AWS IoT Events

AWS IoT Events 可以分析偵測器模型並產生分析結果，而無需將輸入資料傳送至偵測器模型。 AWS IoT Events 會執行本節所述的一系列分析，以檢查您的偵測器模型。此進階故障診斷解決方案也會摘要診斷資訊，包括嚴重性等級和位置，讓您可以快速尋找和修正偵測器模型中的潛在問題。如需偵測器模型診斷錯誤類型和訊息的詳細資訊，請參閱 的偵測器模型分析和診斷資訊 AWS IoT Events。

您可以使用 AWS IoT Events 主控台、API、AWS Command Line Interface (AWS CLI) 或 AWS SDK 來檢視偵測器模型分析的診斷錯誤訊息。

> **Note**
>
> - 您必須修正所有錯誤，才能發佈偵測器模型。
> - 建議您在生產環境中使用偵測器模型之前，先檢閱警告並採取必要的動作。否則，偵測器模型可能無法如預期般運作。
> - 您最多可以同時有 10 個`RUNNING`處於 狀態的分析。

若要了解如何分析偵測器模型，請參閱 分析 AWS IoT Events （主控台） 的偵測器模型或 在 AWS IoT Events (AWS CLI) 中分析偵測器模型。

主題

- 的偵測器模型分析和診斷資訊 AWS IoT Events
- 分析 AWS IoT Events （主控台）的偵測器模型
- 在 AWS IoT Events (AWS CLI) 中分析偵測器模型

的偵測器模型分析和診斷資訊 AWS IoT Events

偵測器模型分析會收集下列診斷資訊：

- 層級 – 分析結果的嚴重性層級。根據嚴重性等級，分析結果分為三個一般類別：
  - 資訊 (INFO) – 資訊結果會告訴您偵測器模型中的重要欄位。這種類型的結果通常不需要立即採取行動。
  - 警告 (WARNING) – 警告結果會特別注意可能導致偵測器模型問題的欄位。建議您在生產環境中使用偵測器模型之前，先檢閱警告並採取必要的動作。否則，偵測器模型可能無法如預期般運作。
  - 錯誤 (ERROR) – 錯誤結果會通知您偵測器模型中發現的問題。當您嘗試發佈偵測器模型時， AWS IoT Events 會自動執行這組分析。您必須修正所有錯誤，才能發佈偵測器模型。
- 位置 – 包含可用來尋找分析結果參考之偵測器模型中 欄位的資訊。位置通常包含狀態名稱、轉換事件名稱、事件名稱和表達式（例如 `in state TemperatureCheck in onEnter in event Init in action setVariable`)。
- 類型 – 分析結果的類型。分析類型分為下列類別：
  - `supported-actions` – AWS IoT Events 可以在偵測到指定的事件或轉換事件時叫用動作。您可以定義內建動作，以使用計時器或設定變數，或將資料傳送至其他 AWS 服務。您必須指定在提供服務 AWS 的 AWS 區域中使用其他 AWS 服務的動作。
  - `service-limits` – 服務配額也稱為限制，是您 AWS 帳戶的服務資源或操作數量上限或下限。除非另有說明，否則每個配額都是區域特定的。根據您的業務需求，您可以更新偵測器模型，以避免遇到限制或請求提高配額。您可以要求提高某些配額，而其他配額無法提高。如需詳細資訊，請參閱 配額。
- **`structure`** – 偵測器模型必須具備所有必要的元件，例如狀態，並遵循 AWS IoT Events 支援的結構。偵測器模型必須至少有一個狀態和評估傳入輸入資料的條件，以偵測重大事件。偵測到事件時，偵測器模型會轉換為下一個狀態，並可以叫用 動作。這些事件稱為轉換事件。轉換事件必須指示下一個狀態才能進入。
- **`expression-syntax`** – AWS IoT Events 在您建立和更新偵測器模型時， 提供多種指定值的方法。您可以在表達式中使用常值、運算子、函數、參考和替換範本。您可以使用表達式來指定常值，也可以在指定特定值之前 AWS IoT Events 評估表達式。您的表達式必須遵循必要的語法。如需詳細資訊，請參閱篩選、轉換和處理事件資料的表達式。

中的偵測器模型表達式 AWS IoT Events 可以參考特定資料或資源。

- **`data-type`** – AWS IoT Events 支援整數、小數、字串和布林資料類型。如果 AWS IoT Events 可以在表達式評估期間自動將一種資料類型的資料轉換為另一種資料類型，則這些資料類型是相容的。

  > **Note**
  > - 整數和小數是 唯一支援的相容資料類型 AWS IoT Events。
  > - AWS IoT Events 無法評估算術表達式，因為 AWS IoT Events 無法將整數轉換為字串。

- **`referenced-data`** – 您必須定義偵測器模型中參考的資料，才能使用資料。例如，如果您想要將資料傳送至 DynamoDB 資料表，您必須先定義參考資料表名稱的變數，才能在表達式 () 中使用變數`$variable.TableName`。
- **`referenced-resource`** – 偵測器模型使用的資源必須可用。您必須先定義資源，才能使用它們。例如，您想要建立偵測器模型來監控溫室的溫度。您必須定義輸入 (`$input.TemperatureInput`)，將傳入的溫度資料路由到偵測器模型，才能使用 `$input.TemperatureInput.sensorData.temperature` 參考溫度。

請參閱下一節以疑難排解錯誤，並從偵測器模型的分析中尋找可能的解決方案。

故障診斷 中的偵測器模型錯誤 AWS IoT Events

上述錯誤類型提供偵測器模型的診斷資訊，並對應至您可能擷取的訊息。使用這些訊息和建議的解決方案來疑難排解偵測器模型的錯誤。

訊息和解決方案

- Location
- supported-actions
- service-limits
- structure
- expression-syntax
- data-type
- referenced-data
- referenced-resource

`Location`

具有 相關資訊的分析結果`Location`對應至下列錯誤訊息：

- 訊息 – 包含有關分析結果的其他資訊。這可以是資訊、警告或錯誤訊息。

  解決方案：如果您指定了 AWS IoT Events 目前不支援的動作，您可能會收到此錯誤訊息。如需支援的動作清單，請參閱 在 中接收資料和觸發動作的支援動作 AWS IoT Events。

`supported-actions`

具有 相關資訊的分析結果`supported-actions`對應於下列錯誤訊息：

- 訊息：動作定義中存在無效的動作類型：*`####`*。

  解決方案：如果您指定了 AWS IoT Events 目前不支援的動作，您可能會收到此錯誤訊息。如需支援的動作清單，請參閱 在 中接收資料和觸發動作的支援動作 AWS IoT Events。

- 訊息： DetectorModel 定義具有 *`aws-service`*動作，但 *`region-name`* 不支援*`aws-service`*此服務。

  解決方案：如果您指定的動作受到支援 AWS IoT Events，但目前區域中無法使用該動作，則可能會收到此錯誤訊息。當您嘗試將資料傳送至 區域中無法使用 AWS 的服務時，可能會發生這種情況。您還必須為 AWS IoT Events 和 AWS 您正在使用的服務選擇相同的區域。

`service-limits`

具有 相關資訊的分析結果`service-limits`對應於下列錯誤訊息：

- 訊息：承載中允許的內容表達式超過 state-name 中 *`event-name`* 中的 *`content-expression-size`* 位元組限制。

  解決方案：如果動作承載的內容表達式大於 1024 個位元組，您可能會收到此錯誤訊息。承載的內容表達式大小最多可達 1024 個位元組。

- 訊息：偵測器模型定義中允許的狀態數量超過*`states-per-detector-model`*。

  解決方案：如果您的偵測器模型有超過 20 個狀態，您可能會收到此錯誤訊息。偵測器模型最多可有 20 個狀態。

- 訊息：計時器*`#####`*的持續時間應至少為*`minimum-timer-duration`*秒長。

解決方案：如果您的計時器持續時間少於 60 秒，您可能會收到此錯誤訊息。我們建議計時器的持續時間介於 60 到 31622400 秒之間。如果您指定計時器持續時間的表達式，持續時間表達式的評估結果會四捨五入至最接近的整數。

- 訊息：每個事件允許的動作數量超過偵測器模型定義中*actions-per-event*限制

  解決方案：如果事件有 10 個以上的動作，您可能會收到此錯誤訊息。對於偵測器模型中的每個事件，您最多可以有 10 個動作。

- 訊息：每個狀態允許的轉換事件數超過偵測器模型定義中每個*transition-events-per-state*state 限制。

  解決方案：如果狀態有 20 個以上的轉換事件，您可能會收到此錯誤訊息。偵測器模型中的每個狀態最多可以有 20 個轉換事件。

- 訊息：每個狀態允許的事件數量超過偵測器模型定義中*events-per-state*數量

  解決方案：如果狀態有 20 個以上的事件，您可能會收到此錯誤訊息。偵測器模型中每個狀態最多可以有 20 個事件。

- 訊息：可與單一輸入相關聯的偵測器模型數量上限可能已達到限制。輸入 *input-name* 用於 *detector-models-per-input* 偵測器模型路由。

  解決方案：如果您嘗試將輸入路由到超過 10 個偵測器模型，您可能會收到此警告訊息。您最多可以有 10 個與單一偵測器模型相關聯的不同偵測器模型。

`structure`

具有 相關資訊的分析結果`structure`對應於下列錯誤訊息：

- 訊息：動作只能定義一種類型，但找到具有*number-of-types*的動作。請分割為個別動作。

  解決方案：如果您使用 API 操作來建立或更新偵測器模型，在單一欄位中指定兩個或多個動作，則可能會收到此錯誤訊息。您可以定義`Action`物件陣列。請務必將每個動作定義為單獨的物件。

- 訊息：TransitionEvent *transition-event-name* 會轉換為不存在的狀態*##*。

  解決方案：如果 AWS IoT Events 找不到轉換事件參考的下一個狀態，您可能會收到此錯誤訊息。請確定已定義下一個狀態，且您已輸入正確的狀態名稱。

- 訊息： DetectorModelDefinition 具有共用狀態名稱：找到*####*，重複*number-of-states*。

解決方案：如果您針對一個或多個狀態使用相同的名稱，您可能會收到此錯誤訊息。請務必為偵測器模型中的每個狀態提供唯一的名稱。狀態名稱必須有 1-128 個字元。有效字元：a-z、A-Z、0-9、_（底線）和 -（連字號）。

- 訊息：定義的 initialStateName *`initial-state-name`* 未對應至定義的狀態。

  解決方案：如果初始狀態名稱不正確，您可能會收到此錯誤訊息。偵測器模型會保持初始（啟動）狀態，直到輸入到達為止。輸入到達後，偵測器模型會立即轉換為下一個狀態。請確定初始狀態名稱是已定義狀態的名稱，而且您輸入正確的名稱。

- 訊息：偵測器模型定義必須在條件中使用至少一個輸入。

  解決方案：如果您未在條件中指定輸入，則可能會收到此錯誤。您必須在至少一個條件中使用至少一個輸入。否則， AWS IoT Events 不會評估傳入的資料。

- 訊息：在 SetTimer 中只能設定秒數和 durationExpression 的其中之一。

  解決方案：如果您同時使用 `seconds`和 `durationExpression`做為計時器，您可能會收到此錯誤訊息。請確定您使用 `seconds`或 `durationExpression`做為 的參數`SetTimerAction`。如需詳細資訊，請參閱 AWS IoT Events API 參考中的 SetTimerAction。

- 訊息：無法連線偵測器模型中的動作。檢查啟動動作的條件。

  解決方案：如果偵測器模型中的動作無法連線，事件的條件會評估為 false。檢查包含 動作的事件條件，以確保其評估為 true。當事件的條件評估為 true 時，動作應該可以連線。

- 訊息：正在讀取輸入屬性，但這可能是計時器過期所造成。

  解決方案：發生下列任一情況時，可以讀取輸入屬性的值：

  - 已收到新的輸入值。
  - 當偵測器中的計時器過期時。

  若要確保只有在收到輸入的新值時才會評估輸入屬性，請在您的 條件中包含對 `triggerType(“Message”)`函數的呼叫，如下所示：

  在偵測器模型中評估的原始條件：

  ```
  if ($input.HeartBeat.status == “OFFLINE”)
  ```

  會變成類似以下內容：

```
if ( triggerType("MESSAGE") &&  $input.HeartBeat.status == “OFFLINE”)
```

其中對triggerType(“Message”)函數的呼叫會在條件中提供的初始輸入之前出現。透過使用此技術，triggerType("Message")函數將評估為 true，並滿足接收新輸入值的條件。如需triggerType函數使用方式的詳細資訊，請在 AWS IoT Events 開發人員指南的表達式區段triggerType中搜尋

- 訊息：無法連線偵測器模型中的狀態。檢查會導致轉換至所需狀態的條件。

  解決方案：如果偵測器模型中的狀態無法連線，則會導致傳入轉換至該狀態的條件會評估為 false。檢查偵測器模型中傳入轉換到該無法連線狀態的條件是否評估為 true，以便達到所需的狀態。

- 訊息：過期計時器可能會導致未預期的訊息數量傳送。

  解決方案：若要防止偵測器模型因計時器過期而進入無限訊息傳送量的無限狀態，請考慮在偵測器模型的條件下使用對triggerType("Message")函數的呼叫，如下所示：

  在偵測器模型中評估的原始條件：

  ```
  if (timeout("awake"))
  ```

  會轉換為如下所示的條件：

  ```
  if (triggerType("MESSAGE") && timeout("awake"))
  ```

  其中對triggerType(“Message”)函數的呼叫會在條件中提供的初始輸入之前出現。

  此變更可防止在偵測器中啟動計時器動作，防止無限循環的訊息傳送。如需如何在偵測器中使用計時器動作的詳細資訊，請參閱《 AWS IoT Events 開發人員指南》中的使用內建動作頁面

expression-syntax

具有 相關資訊的分析結果expression-syntax對應於下列錯誤訊息：

- 訊息：您的承載表達式 {*expression*} 無效。定義的承載類型為 JSON，因此您必須指定 AWS IoT Events 評估為字串的表達式。

  解決方案：如果指定的承載類型是 JSON，則 AWS IoT Events 首先檢查服務是否可以將您的表達式評估為字串。評估的結果不能是布林值或數字。如果驗證不成功，您可能會收到此錯誤。

- 訊息： `SetVariableAction.value` 必須是表達式。無法剖析值 '*variable-value*'

  解決方案：您可以使用 `SetVariableAction`來定義具有 `name`和 的變數`value`。`value` 可以是字串、數字或布林值。您也可以指定 的表達式`value`。如需詳細資訊，請參閱 AWS IoT Events API 參考中的 SetVariableAction。

- 訊息：我們無法剖析 DynamoDB 動作的屬性表達式 (*attribute-name*)。使用正確的語法輸入表達式。

  解決方案：您必須對 `DynamoDBAction`. 替代範本中的所有參數使用表達式。如需詳細資訊，請參閱 AWS IoT Events API 參考中的 DynamoDBAction。

- 訊息：我們無法剖析 DynamoDBv2 動作的 tableName 表達式。使用正確的語法輸入表達式。

  解決方案： `tableName`中的 `DynamoDBv2Action` 必須是字串。您必須使用 的表達式`tableName`。表達式接受文字、運算子、函數、參考和替代範本。如需詳細資訊，請參閱 AWS IoT Events API 參考中的 DynamoDBv2Action。

- 訊息：我們無法將您的表達式評估為有效的 JSON。DynamoDBv2 動作僅支援 JSON 承載類型。

  解決方案： 的承載類型`DynamoDBv2`必須是 JSON。請確定 AWS IoT Events 可以將承載的內容表達式評估為有效的 JSON。如需詳細資訊，請參閱 AWS IoT Events API 參考中的 DynamoDBv2Action。

- 訊息：我們無法剖析 *action-type* 承載的內容表達式。使用正確的語法輸入內容表達式。

  解決方案：內容表達式可以包含字串 ('*string*')、變數 ($variable.*variable-name*)、輸入值 ($input.*input-name*.*path-to-datum*)、字串串連，以及包含 的字串`${}`。

- 訊息：自訂承載必須是非空白的。

  解決方案：如果您為動作選擇自訂承載，但未在 AWS IoT Events 主控台中輸入內容表達式，則可能會收到此錯誤訊息。如果您選擇自訂承載，則必須在自訂承載下輸入內容表達式。如需詳細資訊，請參閱 AWS IoT Events API 參考中的承載。

- 訊息：無法剖析計時器 '*timer-name*' ######## '*duration-expression*'。

  解決方案：計時器持續時間表達式的評估結果必須是介於 60–31622400 之間的值。持續時間的評估結果會四捨五入到最接近的整數。

- 訊息：無法剖析 *action-name* 的表達式 '*expression*'

  解決方案：如果指定動作的表達式語法不正確，您可能會收到此訊息。請務必使用正確的語法輸入表達式。如需詳細資訊，請參閱在 中篩選裝置資料和定義動作的語法 AWS IoT Events。

- 訊息：IotSitewiseAction無法剖析 的 *fieldName*。您必須在表達式中使用正確的語法。

  解決方案：如果 AWS IoT Events 無法剖析 的 *fieldName*，您可能會收到此錯誤IotSitewiseAction。請確定 *fieldName* 使用 AWS IoT Events 可以剖析的表達式。如需詳細資訊，請參閱 AWS IoT Events API 參考中的 IotSiteWiseAction。

data-type

具有 相關資訊的分析結果data-type對應至下列錯誤訊息：

- 訊息：計時器#####的持續時間表達式 *duration-expression* 無效，必須傳回數字。

  解決方案：如果 AWS IoT Events 無法將計時器的持續時間表達式評估為數字，您可能會收到此錯誤訊息。請確定您的 durationExpression 可以轉換為數字。不支援其他資料類型，例如布林值。

- 訊息：運算式 *condition-expression* 不是有效的條件運算式。

  解決方案：如果 AWS IoT Events 無法將 評估condition-expression為布林值，您可能會收到此錯誤訊息。布林值必須為 TRUE或 FALSE。請確定您的條件表達式可以轉換為布林值。如果結果不是布林值，則相當於 FALSE，而且 不會叫用動作或轉換為事件中nextState指定的 。

- 訊息：在下列表達式中找到不相容的資料類型 【*inferred-types*】 以供##： *expression*

  解決方案：解決方案：偵測器模型中相同輸入屬性或變數的所有表達式都必須參考相同的資料類型。

  使用下列資訊來解決問題：

  - 當您使用參考做為具有一或多個運算子的運算元時，請確定您參考的所有資料類型都相容。

    例如，在以下表達式中，整數2是 ==和 &&運算子的運算元。為了確保運算元相容，$variable.testVariable + 1且$variable.testVariable必須參考整數或小數。

    此外，整數1是+運算子的運算元。因此， $variable.testVariable 必須參考整數或小數。

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

  - 當您使用參考做為傳遞給函數的引數時，請確定該函數支援您參考的資料類型。

    例如，下列timeout("*time-name*")函數需要具有雙引號的字串做為引數。如果您使用#####值的參考，則必須參考具有雙引號的字串。

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

> **Note**
>
> 對於 convert(*type*, *expression*)函數，如果您使用##值的參考，則參考的評估結果必須是 String、 Decimal或 Boolean。

如需詳細資訊，請參閱AWS IoT Events 運算式中輸入和變數的參考。

- 訊息：搭配##使用的不相容資料類型 【*inferred-types*】。這可能會導致執行時間錯誤。

  解決方案：如果相同輸入屬性或變數的兩個表達式參考兩種資料類型，您可能會收到此警告訊息。請確定相同輸入屬性或變數的表達式參考偵測器模型中的相同資料類型。

- 訊息：您為運算子 【*###*】 輸入的資料類型 【*####*】 與下列表達式不相容：'*expression*'

  解決方案：如果您的表達式結合與指定運算子不相容的資料類型，您可能會收到此錯誤訊息。例如，在以下表達式中，運算子與整數、十進位和字串資料類型+相容，但不與布林資料類型的運算元相容。

  ```
  true + false
  ```

  您必須確定搭配 運算子使用的資料類型相容。

- 訊息：針對*####*找到的資料類型 【*####*】 不相容，可能導致執行時間錯誤。

  解決方案：如果相同輸入屬性的兩個表達式參考 狀態OnEnterLifecycle的 或 狀態OnExitLifecycle的 OnInputLifecycle和 的兩種資料類型，您可能會收到此錯誤訊息。確定您在 OnEnterLifecycle (或 OnInputLifecycle和 OnExitLifecycle) 中的表達式參考偵測器模型每個狀態的相同資料類型。

- 訊息：承載表達式 【*###*】 無效。指定將在執行時間評估為字串的表達式，因為承載類型是 JSON 格式。

  解決方案：如果您指定的承載類型為 JSON，但 AWS IoT Events 無法將其表達式評估為字串，則可能會收到此錯誤。確定評估的結果是字串，而不是布林值或數字。

- 訊息：您的插入表達式 {*interpolated-expression*} 必須在執行時間評估為整數或布林值。否則，您的承載表達式 {*payload-expression*} 在執行時間將無法剖析為有效的 JSON。

  解決方案：如果 AWS IoT Events 無法將插入表達式評估為整數或布林值，您可能會收到此錯誤訊息。請確定您的插補表達式可以轉換為整數或布林值，因為不支援其他資料類型，例如 tring。

- 訊息：IotSitewiseAction欄位表達式中的###類型定義為類型#####，並推斷為類型####。定義的類型和推斷的類型必須相同。

  解決方案：如果您在 propertyValue 中的表達式IotSitewiseAction具有與推斷的資料類型不同的定義，您可能會收到此錯誤訊息 AWS IoT Events。請務必針對偵測器模型中此表達式的所有執行個體使用相同的資料類型。

- 訊息：用於setTimer動作的資料類型 【*inferred-types*】 不會Integer針對下列表達式評估為：

  解決方案：如果持續時間表達式的推斷資料類型不是整數或小數，您可能會收到此錯誤訊息。請確定您的 durationExpression 可以轉換為數字。不支援其他資料類型，例如布林值和字串。

- 訊息：與比較運算子 【###】 運算元搭配使用的資料類型 【####】 在下列表達式中不相容：

  解決方案：偵測器模型的條件式表達式 (###) 中###運算元的推斷資料類型不相符。運算元必須與偵測器模型所有其他部分中的相符資料類型搭配使用。

> **Tip**
>
> 您可以使用 convert 來變更偵測器模型中表達式的資料類型。如需詳細資訊，請參閱要在 AWS IoT Events 表達式中使用的函數。

referenced-data

具有 相關資訊的分析結果referenced-data對應於下列錯誤訊息：

- 訊息：偵測到中斷的計時器：計時器#####用於表達式，但從未設定。

  解決方案：如果您使用未設定的計時器，可能會收到此錯誤訊息。在表達式中使用計時器之前，您必須先設定計時器。此外，請確定您輸入正確的計時器名稱。

- 訊息：偵測到損壞的變數：變數####用於表達式，但從未設定。

  解決方案：如果您使用未設定的變數，可能會收到此錯誤訊息。您必須先設定變數，才能在表達式中使用它。此外，請確定您輸入正確的變數名稱。

- 訊息：偵測到損壞的變數：變數會先用於表達式，然後再設定為值。

  解決方案：每個變數都必須指派給一個值，才能在表達式中進行評估。在每次使用之前設定變數的值，以便擷取其值。此外，請確定您輸入正確的變數名稱。

`referenced-resource`

具有 相關資訊的分析結果`referenced-resource`對應至下列錯誤訊息：

- 訊息：偵測器模型定義包含參考不存在的輸入。

  解決方案：如果您使用表達式來參考不存在的輸入，您可能會收到此錯誤訊息。請確定您的表達式參考現有的輸入，並輸入正確的輸入名稱。如果您沒有輸入，請先建立一個輸入。

- 訊息：偵測器模型定義包含無效的 InputName： *`input-name`*

  解決方案：如果您的偵測器模型包含無效的輸入名稱，您可能會收到此錯誤訊息。請確定您輸入正確的輸入名稱。輸入名稱必須有 1-128 個字元。有效字元：a-z、A-Z、0-9、_（底線）和 -（連字號）。

分析 AWS IoT Events （主控台）的偵測器模型

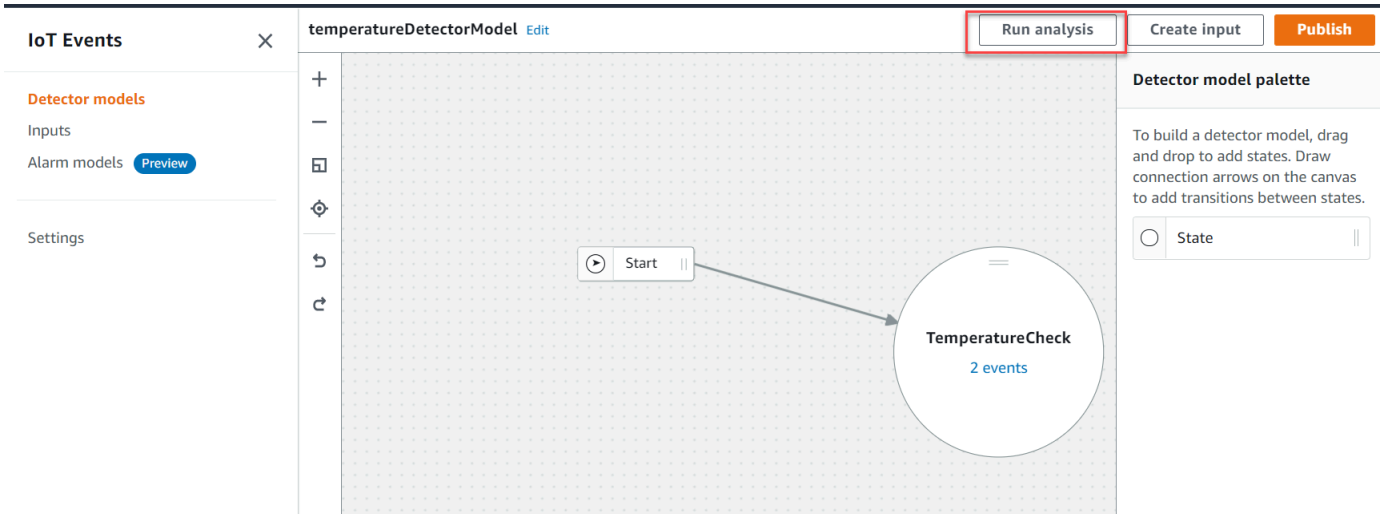
AWS IoT Events 可讓您透過偵測事件並使用 AWS IoT Events API 觸發動作，來監控 IoT 資料並對其做出反應。下列步驟使用 AWS IoT Events 主控台來分析偵測器模型。

> **Note**
>
> AWS IoT Events 開始分析偵測器模型後，您有最多 24 小時的時間擷取分析結果。

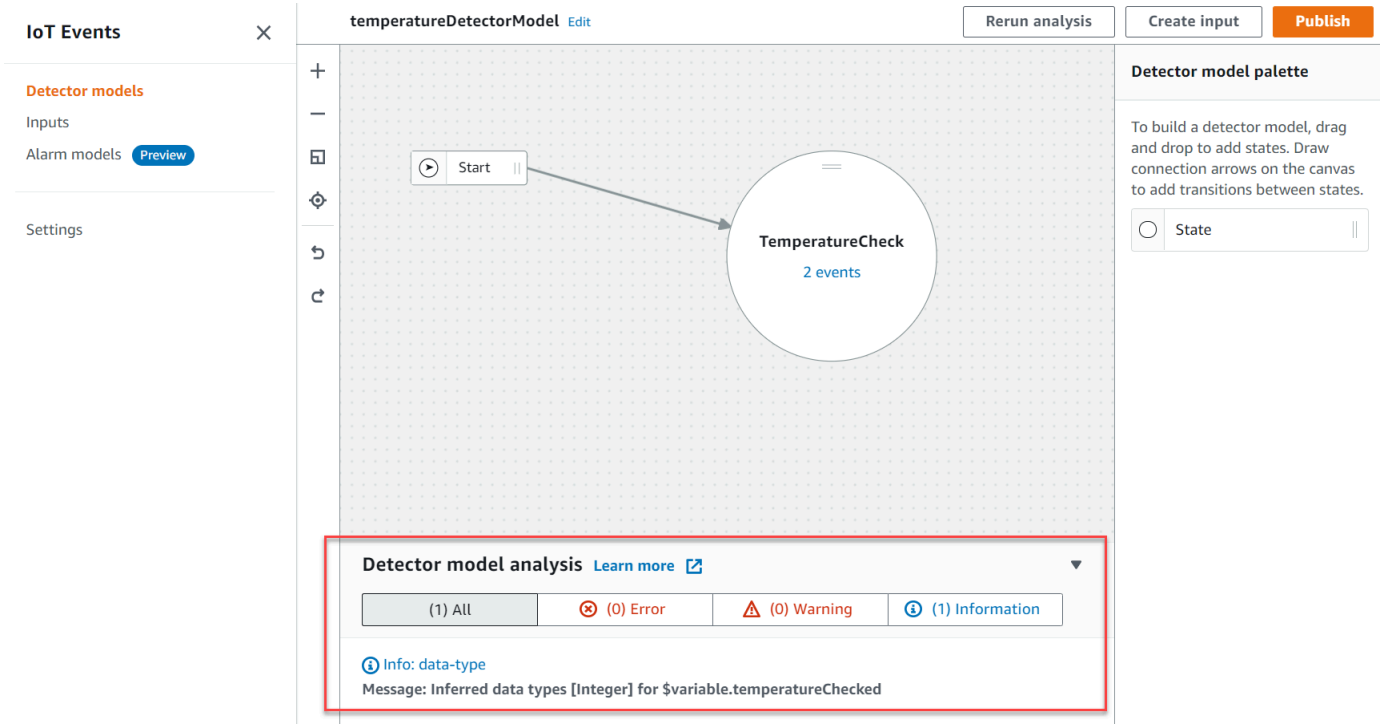
偵測器模型分析可協助您最佳化模型、識別潛在問題，並確保它們如預期般運作。例如，在風力槍上，偵測器模型分析可能會顯示模型是否根據異常震動模式正確識別潛在的齒輪故障。或者，如果模型在風速超過安全操作閾值時準確觸發維護警示。透過根據分析精簡模型，您可以改善預測性維護、減少停機時間，並提高整體能源生產效率。

分析偵測器模型

1. 登入 AWS IoT Events 主控台。
2. 在導覽窗格中，選擇偵測器模型。
3. 在偵測器模型下，選擇目標偵測器模型。
4. 在偵測器模型頁面上，選擇編輯。
5. 在右上角，選擇執行分析。



以下是 AWS IoT Events 主控台中的範例分析結果。



在 AWS IoT Events (AWS CLI) 中分析偵測器模型

以程式設計方式分析您的 AWS IoT Events 偵測器模型，可提供有關其結構、行為和效能的寶貴洞見。此 API 型方法允許自動化分析、與現有工作流程整合，以及跨多個偵測器模型執行大量操作的能力。透過利用 StartDetectorModelAnalysis API，您可以啟動模型的深入檢查、協助您識別潛在問題、最佳化邏輯流程，並確保您的 IoT 事件處理符合您的業務需求。

下列步驟使用 AWS CLI 來分析偵測器模型。

使用 分析偵測器模型 AWS CLI

1. 執行下列命令以開始分析。

```
aws iotevents start-detector-model-analysis --cli-input-json file://file-name.json
```

> **Note**
>
> 將 *file-name* 取代為包含偵測器模型定義的檔案名稱。

Example偵測器模型定義

```
{
    "detectorModelDefinition": {
        "states": [
            {
                "stateName": "TemperatureCheck",
                "onInput": {
                    "events": [
                        {
                            "eventName": "Temperature Received",
                            "condition":
 "isNull($input.TemperatureInput.sensorData.temperature)==false",
                            "actions": [
                                {
                                    "iotTopicPublish": {
                                        "mqttTopic": "IoTEvents/Output"
                                    }
                                }
                            ]
                        }
                    ],
                    "transitionEvents": []
                },
                "onEnter": {
                    "events": [
                        {
                            "eventName": "Init",
                            "condition": "true",
```

```
                            "actions": [
                                {
                                    "setVariable": {
                                        "variableName": "temperatureChecked",
                                        "value": "0"
                                    }
                                }
                            ]
                        }
                    ]
                },
                "onExit": {
                    "events": []
                }
            }
        ],
        "initialStateName": "TemperatureCheck"
    }
}
```

如果您使用 AWS CLI 來分析現有的偵測器模型，請選擇下列其中一項來擷取偵測器模型定義：

- 如果您想要使用 AWS IoT Events 主控台，請執行下列動作：
  1. 在導覽窗格中，選擇偵測器模型。
  2. 在偵測器模型下，選擇目標偵測器模型。
  3. 從動作中選擇匯出偵測器模型，以下載偵測器模型。偵測器模型會儲存在 JSON 中。
  4. 開啟偵測器模型 JSON 檔案。
  5. 您只需要 `detectorModelDefinition` 物件。移除下列項目：
     - 頁面頂端的第一個大括號 ({)
     - 該`detectorModel`行
     - `detectorModelConfiguration` 物件
     - 頁面底部的最後一個大括號 (})
  6. 儲存檔案。
- 如果您想要使用 AWS CLI，請執行下列動作：
  1. 在終端機執行下列命令。

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

2. 將 *detector-model-name* 取代為偵測器模型的名稱。
3. 將detectorModelDefinition物件複製到文字編輯器。
4. 在 之外新增大括號 ({})detectorModelDefinition。
5. 在 JSON 中儲存檔案。

Example回應範例

```
{
    "analysisId": "c1133390-14e3-4204-9a66-31efd92a4fed"
}
```

2. 從輸出複製分析 ID。
3. 執行下列命令來擷取分析的狀態。

```
aws iotevents describe-detector-model-analysis --analysis-id "analysis-id"
```

> **Note**
>
> 將 *analysis-id* 取代為您複製的分析 ID。

Example回應範例

```
{
    "status": "COMPLETE"
}
```

狀態可以是下列其中一個值：

- RUNNING – AWS IoT Events 正在分析偵測器模型。此程序最多可能需要一分鐘才能完成。
- COMPLETE – AWS IoT Events 已完成分析偵測器模型。
- FAILED – AWS IoT Events 無法分析偵測器模型。請稍後再試。

4. 執行下列命令來擷取偵測器模型的一或多個分析結果。

> **Note**
>
> 將 *analysis-id* 取代為您複製的分析 ID。

```
aws iotevents get-detector-model-analysis-results --analysis-id "analysis-id"
```

Example回應範例

```
{
    "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 開始分析偵測器模型後，您有最多 24 小時的時間擷取分析結果。

AWS IoT Events 命令

本章提供 中所有可用 API 操作的完整指南 AWS IoT Events。它提供詳細說明，包括範例請求、回應，以及支援 Web 服務通訊協定中每個操作的潛在錯誤。了解這些 API 操作可協助您有效地 AWS IoT Events 整合至 IoT 應用程式，並自動化事件偵測和回應工作流程。

AWS IoT Events 動作

您可以使用 AWS IoT Events API 命令來建立、讀取、更新和刪除輸入和偵測器模型，以及列出其版本。如需詳細資訊，請參閱 AWS IoT Events API 參考 AWS IoT Events 中 支援的動作和資料類型。

AWS CLI 命令參考中的AWS IoT Events 各節包含您可以用來管理和操作的 AWS CLI 命令 AWS IoT Events。

AWS IoT Events 資料

您可以使用 AWS IoT Events Data API 命令將輸入傳送到偵測器、列出偵測器，以及檢視或更新偵測器的狀態。如需詳細資訊，請參閱 AWS IoT Events API 參考中 AWS IoT Events 資料支援的動作和資料類型。

AWS CLI 命令參考中的AWS IoT Events 資料區段包含您可以用來處理 AWS IoT Events 資料的 AWS CLI 命令。

的文件歷史記錄 AWS IoT Events

下表說明 2020 年 9 月 17 日之後 AWS IoT Events 開發人員指南的重要變更。如需有關此文件更新的詳細資訊，您可以訂閱 RSS 摘要。

變更	描述	日期
終止支援通知	終止支援通知：2026 年 5 月 20 日， AWS 將停止對 的支援 AWS IoT Events。2026 年 5 月 20 日之後，您將無法再存取 AWS IoT Events 主控台或 AWS IoT Events 資源。	2025 年 5 月 20 日
區域啟動	AWS IoT Events 現可於亞太區域（孟買）區域使用。	2021 年 9 月 30 日
區域啟動	AWS IoT Events 現在可在 AWS GovCloud（美國西部）區域使用。	2021 年 9 月 22 日
透過執行分析對偵測器模型進行故障診斷	AWS IoT Events 現在可以分析偵測器模型，並產生分析結果，供您用來疑難排解偵測器模型。	2021 年 2 月 23 日
區域啟動	AWS IoT Events 在中國（北京）推出。	2020 年 9 月 30 日
表達式用量	新增範例，示範如何撰寫表達式。	2020 年 9 月 22 日
使用警示進行監控	警示可協助您監控資料是否有變更。您可以建立警示，在超過閾值時傳送通知。	2020 年 6 月 1 日

舊版更新

下表說明 AWS IoT Events 開發人員指南在 2020 年 9 月 18 日之前的重要變更。

變更	描述	日期
已將類型驗證新增至運算式參考	已將類型驗證資訊新增至運算式參考。	2020 年 8 月 3 日
新增其他服務的區域警告	新增有關為 AWS IoT Events 和其他 AWS 服務選取相同區域的警告。	2020 年 5 月 7 日
新增、更新	• 承載自訂功能 • 新的事件動作：Amazon DynamoDB 和 AWS IoT SiteWise	2020 年 4 月 27 日
新增偵測器模型條件表達式的內建函數	新增偵測器模型條件表達式的內建函數。	2019 年 9 月 10 日
新增偵測器模型範例	新增偵測器模型的範例。	2019 年 8 月 5 日
新增事件動作	已為 新增新事件動作： • Lambda • Amazon SQS • Kinesis Data Firehose • AWS IoT Events 輸入	2019 年 7 月 19 日
新增、更正	• 已更新 `timeout()` 函數的描述。 • 新增有關帳戶閒置的最佳實務。	2019 年 6 月 11 日
已更新許可政策和主控台偵錯選項	• 已更新主控台許可政策。 • 已更新主控台偵錯選項頁面映像。	2019 年 6 月 5 日

變更	描述	日期
更新	AWS IoT Events 服務開放一般可用性。	2019 年 5 月 30 日
新增、更新	• 更新安全性資訊。 • 新增了註釋偵測器模型範例。	2019 年 5 月 22 日
新增範例和必要的許可	新增 Amazon SNS 承載範例；新增 的必要許可`CreateDetectorModel` 。	2019 年 5 月 17 日
新增其他安全性資訊	已將資訊新增至安全性章節。	2019 年 5 月 9 日
有限的預覽版本	文件的有限預覽版本。	2019 年 3 月 28 日