開發人員指南

AWS IoT Events

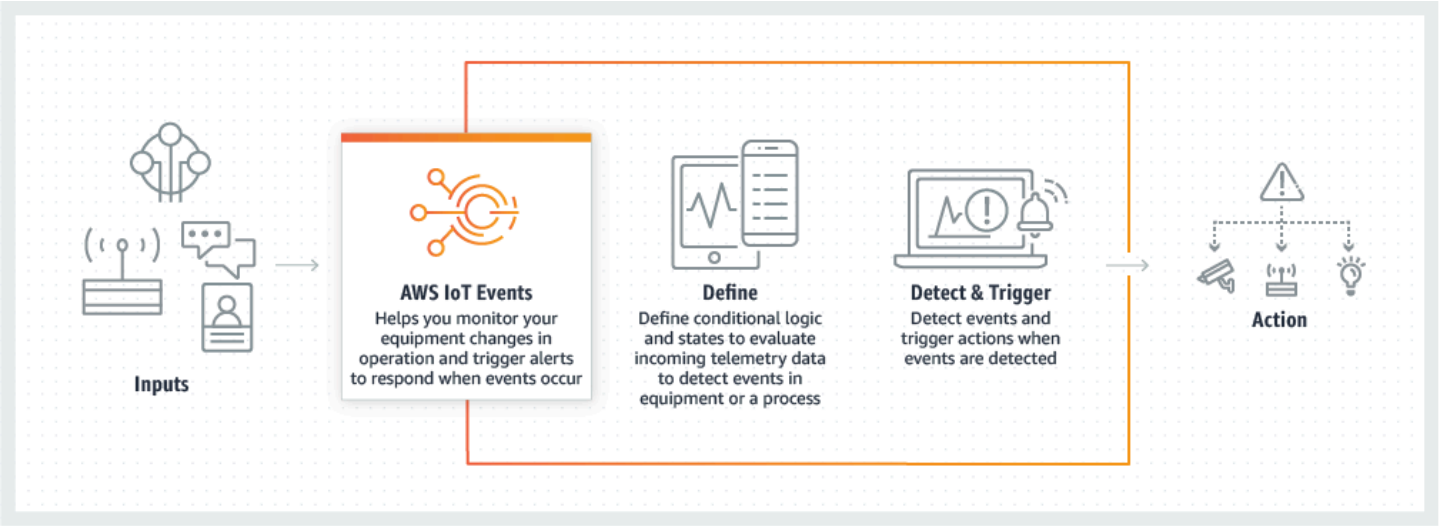
AWS IoT Events: 開發人員指南

Table of Contents

什麼是 AWS IoT Events？

AWS IoT Events 可讓您監控設備或裝置叢集是否發生故障或作業變更，並在此類事件發生時觸發動作。 AWS IoT Events 持續監視來自裝置、程序、應用程式和其他 AWS 服務的 IoT 感測器資料，以識別重要事件，以便您採取行動。

您可以使用 AWS IoT Events 在 AWS Cloud 中建置複雜的事件監控應用程式，以便透過 AWS IoT Events 主控台或 API 存取這些應用程式。



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。其中包括感測器裝置、管理應用程式，以及其他 AWS IoT 服務，例如 AWS IoT Core 和 AWS IoT Analytics。您可以使用標準 API 介面 (`BatchPutMessage`API) 將 AWS IoT Events 任何遙測資料輸入推送至。

使用簡單的邏輯表達式來識別事件的複雜模式

AWS IoT Events 可以識別涉及來自單個 IoT 設備或應用程序的多個輸入的事件模式，或來自多種設備和許多獨立傳感器。這特別有用，因為每個傳感器和應用程序都提供了重要信息。但是，只有結合不同的感測器和應用程式資料，才能全面掌握操作效能和品質。您可以設定 AWS IoT Events 偵測器，使用簡單的邏輯運算式來辨識這些事件，而不是複雜的程式碼。

根據事件觸發動作

AWS IoT Events 可讓您直接在 Amazon Simple Notification Service (Amazon SNS)、Lambda AWS IoT Core、Amazon SQS 和亞馬 Amazon Kinesis Firehose 中觸發動作。您也可以使用 AWS

IoT 規則引擎觸發 AWS Lambda 功能，以便使用其他服務 (例如 Amazon Connect) 或您自己的企業資源規劃 (ERP) 應用程式來執行動作。

AWS IoT Events 包含您可以執行的動作的預先建置程式庫，也可讓您定義自己的動作。

自動擴充以滿足您車隊的需求

AWS IoT Events 當您連接同質裝置時，會自動縮放。您可以為特定類型的裝置定義一次偵測器，服務會自動擴展和管理連線到該裝置的所有執行個體 AWS IoT Events。

使用案例

監控和維護遠端裝置

您需要監視遠端部署的機器群。如果其中一個停止運作，而您沒有其他導致失敗的內容，您可能必須立即更換整個處理單元或機器。但這是不可持續的。使用時， AWS IoT Events 您可以從每台機器上的多個傳感器接收消息，並通過使用隨時間發送的錯誤代碼診斷確切的問題。現在，您無需更換所有內容，而是獲得了所需的信息，只需將需要更換的部件發送給技術人員。擁有數百萬台機器，節省成本最多可達數百萬美元，從而降低擁有或維護每台機器的總成本。

管理工業機器人

您可以在設施內部署機器人，以自動化套件的移動。為了將機器人的成本降到最低，機器人配備了簡單，低成本的傳感器，可以向雲端報告信息。但是您的機器人具有數十種傳感器和數百種操作模式，因此很難在發生問題時發現問題。您可以使用建置專家系統 AWS IoT Events，在雲端處理感應器資料，並建立警示，以便在即將發生故障時自動警告技術人員。

追蹤樓宇自動化系統

您可以操作大量資料中心，這些資料中心必須受到高溫和低濕度監控，以防止違反這些環境閾值時發生設備故障。您使用的感測器是向許多製造商購買的，每種類型都有自己的管理軟體。但是，來自不同廠商的管理軟體並不相容，因此很難偵測到問題。使用時 AWS IoT Events，您可以設定警示，在發生故障之前，通知營運分析師有關加熱與冷卻系統的問題。透過這種方式，您可以防止資料中心意外關機，這會花費數千美元的設備更換和潛在的收入損失。

設定 AWS IoT Events

如果您沒有 AWS 帳戶，請完成以下步驟來建立一個。

若要註冊成為 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，並在電話鍵盤輸入驗證碼。

   當您註冊一個時 AWS 帳戶，將創建AWS 帳戶根使用者一個。根使用者有權存取該帳戶中的所有 AWS 服務 和資源。作為最佳安全實務，將管理存取權指派給管理使用者，並且僅使用根使用者來執行需要根使用者存取權的任務。

設定的權限 AWS IoT Events

本節說明使用的某些功能所需的角色和權限 AWS IoT Events。您可以使用 AWS CLI 命令或 AWS Identity and Access Management (IAM) 主控台建立角色和相關的權限政策，以存取中的資源或執行特定功能 AWS IoT Events。

IAM 使用者指南提供更多有關安全控制存取 AWS 資源許可的詳細資訊。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 角色跨 AWS 帳戶委派存取權限。

> **Note**
>
> 按鍵可以是 1-128 個字元，可包括：
>
> - 大寫或小寫字母 a-z
> - 數字, 0-9
> - 特殊字元 -、_ 或：。

動作權限

AWS IoT Events 可讓您觸發使用其他 AWS 服務的動作。若要這麼做，您必須 AWS IoT Events 授與代表您執行這些動作的權限。本節包含動作清單，以及授與對資源執行所有這些動作之權限的範例政策。視需要變更*##*和*## ID* 參照。如果可能的話，您也應該變更萬用字元 (*)，以參照將要存取的特定資源。您可以使用 IAM 主控台授與傳 AWS IoT Events 送已定義之 Amazon SNS 警示的權限。

AWS IoT Events 支援下列可讓您使用計時器或設定變數的動作：

- `setTimer`創建一個計時器。
- `resetTimer`以重設計時器。
- `clearTimer`刪除計時器。
- `setVariable`以建立變數。

AWS IoT Events 支援下列可讓您使用 AWS 服務的動作：

- `iotTopicPublish`以發佈有關 MQTT 主題的訊息。
- `iotEvents`將資料 AWS IoT Events 作為輸入值傳送至。
- `iotSiteWise` 將資料傳送至 AWS IoT SiteWise 中的資產屬性。
- `dynamoDB`將資料傳送至亞 Amazon DynamoDB 資料表。
- `dynamoDBv2`將資料傳送至亞 Amazon DynamoDB 資料表。
- `firehose`將數據發送到 Amazon 數據 Firehose 流。
- `lambda`調用一個 AWS Lambda 函數。
- `sns`以推送通知的形式傳送資料。
- `sqs`將資料傳送到 Amazon SQS 佇列。

Example 政策

```
{
   "Version": "2012-10-17",
   "Statement": [
     {
       "Effect": "Allow",
       "Action": "iot:Publish",
       "Resource": "arn:aws:iot:<region>:<account_id>:topic/*"
     },
```

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

保護輸入資料

重要的是要考慮誰可以授予對輸入數據的訪問權限，以便在檢測器模型中使用。如果您有要限制其整體權限的使用者或實體，但該使用者或實體可以建立或更新偵測器模型，則您還必須授予該使用者或實體更新輸入路由的權限。這表示除了授與權限之外`iotevents:CreateDetectorModel``iotevents:UpdateDetectorModel`，您還必須授與的權限`iotevents:UpdateInputRouting`。

Example

下列原則會新增的權限`iotevents:UpdateInputRouting`。

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

您可以為 "" 指定輸入 Amazon 資源名稱 (ARN) 的清單，而不是萬用字元 * "`Resource`"，以將此權限限制為特定輸入。這可讓您限制對由使用者或實體建立或更新的偵測器模型所使用之輸入資料的存取。

Amazon CloudWatch 日誌角色政策

下列原則文件提供角色原則和信任原則，可 AWS IoT Events 讓您代表您提 CloudWatch 交記錄檔。

角色政策：

```
{
    "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：

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

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

您還需要將 IAM 許可政策附加到使用者，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》中的授與使用者將角色傳遞給 AWS 服務的權限。

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

```
            "iam:GetRole",
            "iam:PassRole"
        ],
        "Resource": "arn:aws:iam::<account-id>:role/Role_To_Pass"
    }
  ]
}
```

您可以使用以下命令來放置 CloudWatch 日誌的資源策略。這允許 AWS IoT Events 將日誌事件放入 CloudWatch 流中。

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

使用下面的命令來把日誌記錄選項。以您建立`roleArn`的記錄角色取代。

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

Amazon SNS 簡訊角色政策

下列原則文件提供允許 AWS IoT Events 傳送 SNS 訊息的角色原則和信任原則。

角色政策：

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

```
    ]
}
```

信任政策：

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

開始使用主 AWS IoT Events 控台

本節介紹如何使用控制台創建輸入和檢測AWS IoT Events 器模型。您可以為引擎的兩種狀態建立模型：正常狀態和過壓情況。當引擎中測得的壓力超過特定閾值時，模型會從正常狀態轉變為過壓狀態。然後它會傳送 Amazon SNS 訊息，提醒技術人員有關情況的資訊。當壓力再次低於連續三次壓力讀數的臨界值時，模型會返回正常狀態，並傳送另一則 Amazon SNS 訊息做為確認訊息。

我們檢查低於壓力閾值的連續三個讀數，以消除在非線性恢復階段或異常壓力讀數的情況下可能出現過壓或正常消息的口吃。

在控制台上，您還可以找到幾個可以自定義的預製檢測器模型模板。您還可以使用控制台導入其他人編寫的檢測器模型或導出您的檢測器模型，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必要的輸入或重新建立這些輸入，並更新使用的任何角色 ARN。

在控制台上，您還可以找到幾個可以自定義的預製檢測器模型模板。您也可以使用控制台導入其他人編寫的檢測器模型或導出您的檢測器模型，並在其他模型中使用它們 AWS 區域。如果您匯入偵測器模型，請確定您為新區域建立必要的輸入或重新建立這些輸入，並更新使用的任何角色 ARN。

使用 AWS IoT Events 控制台了解以下內容。

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。這是通過將消息作為輸入發送到完成的 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，為將 AWS IoT Events訊息資料轉寄至的 AWS IoT 規則引擎撰寫AWS IoT Events 動作規則。您必須按名稱識別輸入。
- 在中 AWS IoT Analytics，使用 CreateDataset作業建立資料集`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，裝置才能以這種方式傳送資料。若要這麼做，請為每個輸入指定名稱，並指定輸入監視的內送訊息資料中的哪些欄位。

建立偵測器模型

使用狀態定義檢測器模型（設備或過程的模型）。針對每個狀態，定義評估傳入輸入以偵測重要事件的條件式 (Boolean) 邏輯。當檢測器模型檢測到事件時，它可以使用其他 AWS 服務更改狀態或啟動自定義或預定義的操作。您可以定義其他事件，這些事件會在進入或退出狀態時啟動動作，以及在符合條件時 (選擇性)。

在本教學中，您會傳送 Amazon SNS 訊息做為模型進入或結束特定狀態時的動作。

監視裝置或程序

如果您監視多個裝置或程序，請在每個輸入中指定一個欄位，以識別輸入來源的特定裝置或程序。請參閱中的key欄位CreateDetectorModel。當由標識的輸入字段key識別一個新的值，一個新的設備被識別，並創建一個檢測器。每個檢測器都是檢測器模型的一個實例。新的檢測器繼續響應來自該設備的輸入，直到其檢測器型號被更新或刪除。

如果您監視單一處理序 (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。在這種情況下，當第一個輸入到達時，模型創建一個檢測器（實例）。

將消息作為輸入發送到您的檢測器模型

有幾種方法可以將消息作為輸入從設備或進程發送到 AWS IoT Events 檢測器中，而不需要您對消息執行其他格式化。在本教學課程中，您會使用 AWS IoT 主控台為將 AWS IoT Events郵件資料轉寄至的 AWS IoT 規則引擎撰寫AWS IoT Events 動作規則。

若要這麼做，請依名稱識別輸入，並繼續使用 AWS IoT 主控台來產生作為輸入轉寄至的訊息 AWS IoT Events。

> **Note**
>
> 本教程使用控制台創建相同的控制台，input並在detector model示例中顯示教學課程。您可以使用此 JSON 範例來協助您遵循教學課程。

主題

必要條件

如果您沒有 AWS 帳戶，請創建一個。

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，並在電話鍵盤輸入驗證碼。

   當您註冊一個時 AWS 帳戶，將創建AWS 帳戶根使用者一個。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，將管理存取權指派給管理使用者，並且僅使用根使用者來執行 需要根使用者存取權的任務。

3. 建立兩個 Amazon Simple Notification Service (Amazon SNS) 主題。

   本教學課程 (以及對應的範例) 假設您建立了兩個 Amazon SNS 主題。這些主題的 ARN 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 主題的 ARN。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 主題之外，您還可以讓偵測器傳送 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 主控台訂閱並監視傳送至這些 MQTT 主題的訊息，以確認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警示。您也可以使用在偵測器模型中建立的輸入或變數，在執行階段動態定義 MQTT 主題名稱。

4. 選擇一 AWS 區域 個支持 AWS IoT Events。如需詳細資訊，請參閱 AWS 一般參考 中的 AWS IoT Events。如需說明，請參閱使用入門 AWS Management Console中的 AWS Management Console。

建立輸入

當您為模型建構輸入時，我們建議您收集包含裝置或程序傳送的範例訊息承載的檔案，以報告其健康狀態。擁有這些文件可幫助您定義所需的輸入。

您可以透過本節所述的多種方法建立輸入。

若要開始使用，請`input.json`在您的本機檔案系統上建立具有下列內容的檔案：

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

現在您有了這個入門input.json文件，您可以創建一個輸入。請使用本節中的其中一個主題，以取得有關使用導覽窗格或使用偵測器模型建立輸入的指示。

主題

在導航窗格中創建輸入

本主題說明如何透過導覽窗格為警示模型或偵測器模型建立輸入。

1. 登入主AWS IoT Events 控台或選取 [建立新 AWS IoT Events 帳戶] 選項。
2. 在主 AWS IoT Events 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. 在主控台右上角，選擇 [建立輸入]。
5. 對於輸入，請輸入可選描述 InputName，然後選擇「上傳檔案」。在顯示的對話方塊中，選取您在概述中建立以建立輸入的input.json檔案。
6. 對於「選擇輸入屬性」，請選取要使用的屬性，然後選擇建立。在此示例中，我們選擇了電動機和傳感器數據。壓力。

在檢測器模型中創建輸入

本主題說明如何定義偵測器模型的輸入，以接收遙測資料或訊息。

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 Create new (建立新項目)。
4. 選擇 Create input (建立輸入)。
5. 對於輸入，請輸入可選描述 InputName，然後選擇「上傳檔案」。在顯示的對話方塊中，選取您在概述中建立以建立輸入的input.json檔案。
6. 對於「選擇輸入屬性」，請選取要使用的屬性，然後選擇建立。在此示例中，我們選擇了電動機和傳感器數據。壓力。

建立偵測器模型

在本主題中，您將使用狀態定義偵測器模型 (設備或製程的模型)。

對於每個狀態，您可以定義條件式 (布林值) 邏輯來評估傳入輸入以偵測重要事件。偵測到事件時，它會變更狀態並啟動其他動作。這些事件稱為轉換事件。

在您的狀態中，您還可以定義事件，這些事件可以在檢測器進入或退出該狀態或收到輸入時（這些被稱為`OnEnter``OnExit`和`OnInput`事件）時運行操作。只有當事件的條件邏輯評估為時，動作才會執行`true`。

建立偵測器模型

1. 已為您建立第一個偵測器狀態。要修改它，請在主編輯空間中選擇帶有標籤狀態 _1 的圓。
2. 在 [狀態] 窗格中，輸入 [市/縣] 名稱 OnEnter，然後選擇 [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，enter `true` 表示在輸入狀態時始終啟動事件。
4. 在事件動作下，選擇新增動作。
5. 在「事件動作」 下，執行下列操作：

   a. 選擇設定變數

   b. 對於「變數」作業，選擇「指派值」。

   c. 在變數名稱中，輸入要設定的變數名稱。

   d. 在「變數值」中，輸入值 **`0`** (零)。
6. 選擇儲存。

   像您定義的變量一樣，可以在檢測器模型中的任何事件中設置（給定一個值）。只有在偵測器到達狀態並執行定義或設定動作之後，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」，然後將其拖曳至主編輯空間。這會建立一個標題為的狀態`untitled_state_1`。
9. 暫停在第一個狀態（正常）。狀態的圓周上會出現一個箭頭。
10. 按一下並將箭頭從第一個狀態拖曳到第二個狀態。會顯示從第一個狀態到第二個狀態 (標示為「未命名」) 的直接線。
11. 選取「未命名」的線條。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 [新增轉場事件動作] 窗格上，選擇 [新增動作]。
14. 針對 [選擇動作]，選擇 [設定變數]。
    a. 對於「變數」作業，選擇「指派值」。
    b. 在「變數名稱」中，輸入變數的名稱。
    c. 對於指定值，請輸入值，例如：`$variable.pressureThresholdBreached + 3`
    d. 選擇儲存。
15. 選取未標題為 _state_1 的第二個狀態。
16. 在 [狀態] 窗格中，輸入 [狀態] 名稱，並在 [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. 對於 SNS 主題，請輸入您的 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. 對於 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. 對於 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. 對於 OnExit，選擇新增事件，然後使用您建立的 Amazon SNS 主題的 ARN 輸入並儲存以下事件資訊。

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

20. 暫停在第二狀態（危險）。狀態的圓周上會出現一個箭頭
21. 按一下並將箭頭從第二個狀態拖曳到第一個狀態。隨即出現帶有標籤「未命名」的直線。
22. 選擇「無標題」行，然後在「轉移」事件窗格中，使用下列資訊輸入「事件名稱」和「事件觸發器」邏輯。

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

如需有關為何測試觸發器邏輯中$input值和$variable值的詳細資訊，請參閱中的變數值可用性項目偵測器型號限制和限制。

23. 選取 [開始] 狀態。默認情況下，此狀態是在您創建檢測器模型時創建的）。在「開始」窗格中，選擇「目的地」狀態 (例如，「正常」)。
24. 接下來，配置檢測器模型以監聽輸入。在右上角，選擇「發佈」。
25. 在 [發行偵測器模型] 頁面上，執行下列動作。
    a. 輸入偵測器型號名稱、說明和角色名稱。此角色是為您建立的。
    b. 選擇為每個唯一索引鍵值建立偵測器。若要建立和使用您自己的「角色」，請按照中的步驟進行操作，設定的權限 AWS IoT Events並在此輸入「角色」。
26. 對於「偵測器」建立金鑰，請選擇您先前定義之輸入的其中一個屬性名稱。您選擇作為檢測器創建密鑰的屬性必須存在於每個消息輸入中，並且對於發送消息的每個設備都必須是唯一的。本範例使用 motorid 屬性。
27. 選擇 Save and Publish (儲存並發佈)。

> **Note**
>
> 為給定檢測器模型創建的唯一檢測器的數量基於發送的輸入消息。創建檢測器模型時，將從輸入屬性中選擇一個密鑰。此按鍵決定要使用哪個偵測器執行個體。如果之前沒有看到密鑰（對於此檢測器模型），則會創建一個新的檢測器實例。如果密鑰之前已經看到過，我們使用對應於這個鍵值的現有檢測器實例。

您可以製作檢測器模型定義的備份副本（以 JSON 格式）重新創建或更新檢測器模型，或用作模板以創建另一個檢測器模型。

您可以從控制台或使用以下 CLI 命令執行此操作。如有必要，請變更偵測器模型的名稱，以符合您在上一步中發佈該模型時所使用的名稱。

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

這將創建一個文件（motorDetectorModel.json），其內容類似於以下內容。

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

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

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

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

發送輸入以測試檢測器模型

有幾種方法可以在 AWS IoT Events (請參閱支援的動作) 中接收遙測資料。本主題說明如何在主 AWS IoT 控台中建立 AWS IoT 規則，將訊息作為輸入轉送至 AWS IoT Events 偵測器。您可以使用 AWS IoT 主控台的 MQTT 用戶端傳送測試訊息。當您的裝置能夠使用訊息代理程式傳送 MQTT AWS IoT 訊息 AWS IoT Events 時，您可以使用此方法取得遙測資料。

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「郵件路由」，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。

      - 規則說明。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。從清單中選取適當的選項。
      - SQL 陳述式。輸入 **`SELECT *, topic(2) as motorid FROM 'motors/+/status'`**。

      選擇下一步。
   3. 步驟 3. 附加規則動作。在 「規則動作」區段中，完成下列操作：
      - 動作 1. 選取 IoT Events。會出現下列欄位：

        a. 輸入名稱。從清單中選取適當的選項。如果沒有顯示您的輸入，請選擇「重新整理」。

        若要建立新輸入，請選擇 [建立 IoT Events] 輸入。完成下列欄位：

        - 輸入名稱。輸入 `PressureInput`。
        - 描述。這是選用的。
        - 上傳一個 JSON 文件。上傳 JSON 檔案的複本。如果您沒有文件，則此屏幕上有一個指向示例文件的鏈接。該代碼包括：

```
{
  "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. Batch 模式。這是選用的。如果承載是訊息陣列，請選取此選項。

c. 訊息識別碼。此為選用操作，但建議您採用。

d. IAM 角色。從清單中選取適當的角色。如果未列出角色，請選擇 [建立新角色]。

輸入「角色」名稱並選擇「建立」。

若要新增其他規則，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」，然後從清單中選取適當的動作。

完成顯示的欄位。

- 選擇下一步。

4. 步驟 4. 檢閱並建立。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。輸入用來識別郵件的名稱，例如`motors/Fulton-A32/status`。
- 消息有效負載。輸入下列資料：

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

> **Note**
>
> `messageId`每次發佈新郵件時都會變更。

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

您建立的偵測器執行個體會產生 Amazon SNS 訊息並傳送給您。繼續傳送壓力讀數高於或低於壓力閾值（本範例為 70）的訊息，以查看偵測器正在運作中。

在此範例中，您必須傳送三個壓力讀數低於閾值的訊息，才能轉換回「正常」狀態，並收到 Amazon SNS 訊息，指出超壓情況已清除。回到「正常」狀態後，一則壓力讀數超過限制的訊息會導致偵測器進入「危險」狀態，並傳送指示該狀況的 Amazon SNS 訊息。

現在，您已經創建了一個簡單的輸入和檢測器模型，請嘗試以下操作。

- 在控制台上查看更多檢測器模型示例（模板）。
- 請遵循中簡單的 step-by-step 例子的步驟，使用 AWS CLI
- 瞭解事件中表達式使用的詳細資訊。
- 了解 支援的動作。
- 如果某些東西不起作用，請參閱AWS IoT Events 疑難排解。

的最佳做法 AWS IoT Events

遵循這些最佳做法，以獲得最大的利益 AWS IoT Events。

主題

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

Amazon 會即時 CloudWatch 監控您的 AWS 資源和執行 AWS 的應用程式。您可以透過 CloudWatch 全系統掌握資源使用、應用程式效能和營運狀態。當您開發或偵錯偵 AWS IoT Events 測器模型時，可 CloudWatch協助您瞭解 AWS IoT Events 正在做什麼，以及遇到的任何錯誤。

若要啟用 CloudWatch

1. 如果您尚未執行，請依照中設定的權限 AWS IoT Events的步驟建立具有附加原則的角色，以授與建立和管理 CloudWatch 記錄檔的權限 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. 針對詳細程度等級，選取一個選項。
    b. 對於選取角色，請選取具有足夠權限的角色，以執行您選擇的記錄動作。
    c. (選擇性) 如果您為詳細程度層級選擇除錯，您可以執行下列動作來新增「偵錯」目標：
        i. 在除錯目標下，選擇新增模型選項。
        ii. 輸入偵測器型號名稱和 (選用) KeyValue以指定要記錄的偵測器型號和特定偵測器 (執行個體)。
6. 選擇更新。

您的記錄選項已成功更新。

在 AWS IoT Events 主控台中工作時，定期發佈以儲存偵測器模型

使用 AWS IoT Events 主控台時，正在進行的工作會儲存在本機瀏覽器中。但是，您必須選擇「發佈」，才能將偵測器模型儲存至 AWS IoT Events。發佈偵測器模型後，您發佈的著作將會在您用來存取帳戶的任何瀏覽器中使用。

> **Note**
>
> 如果您未發佈作品，則不會儲存該作品。發佈偵測器模型之後，就無法變更其名稱。但是，您可以繼續修改其定義。

存儲您的 AWS IoT Events 數據，以避免由於長時間不活動而導致數據丟失

如果您沒有使 AWS IoT Events 用很長一段時間，您的數據（包括檢測器模型）可能會自動刪除。例如，很長一段時間可能意味著您不會產生費用，也不會創建檢測器模型。但是，如果沒有至少提前 30 天通知您，我們不會刪除數據或檢測器模型。如果您需要長時間儲存資料，請考慮使用AWS 儲存服務。

教學課程

本章說明如何：

- 取得協助以決定要包含在偵測器模型中的狀態，並判斷您是否需要一個偵測器執行個體或多個偵測器執行個體。
- 請遵循使用AWS CLI.
- 建立輸入以接收來自裝置的遙測資料和偵測器模型，以監視和報告傳送該資料之裝置的狀態。
- 檢閱輸入、偵測器型號和AWS IoT Events服務的限制和限制。
- 查看檢測器模型的更複雜的示例，其中包含註釋。

主題

用AWS IoT Events來監控您的 IoT 裝置

您可以用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執行：

建立輸入

您必須有方法讓您的裝置和程序取得遙測資料AWS IoT Events。您可以透過將訊息做為輸入傳送至AWS IoT Events。您可以通過幾種方式將消息作為輸入發送：

- 使用 BatchPutMessage操作。
- 定義`iotEvents`規則引擎的規則動作。AWS IoT Core規則動作會將訊息資料從您的輸入轉寄至。AWS IoT Events
- 在中AWS IoT Analytics，使用CreateDataset作業建立資料集`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偵測器模型或事`onInput`件中定義 IOTEVentts 動作。`onExit` `transitionEvents`有關偵測器模型執行個體和啟動動作的事件的資訊，會以您指定名稱的輸入方式反饋至系統。

在您的裝置開始以這種方式傳送資料之前，您必須先定義一或多個輸入。若要這麼做，請為每個輸入指定名稱，並指定輸入監視的內送訊息資料中的哪些欄位。 AWS IoT Events以 JSON 有效負載的形式接收來自許多來源的輸入。每個輸入都可以單獨執行操作，也可以與其他輸入結合以檢測更複雜的事件。

建立偵測器模型

使用狀態定義檢測器模型（設備或過程的模型）。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，它可以變更狀態，或使用其他AWS服務啟動自訂或預先定義的動作。您可以定義其他事件，這些事件會在進入或退出狀態時啟動動作，以及在符合條件時 (選擇性)。

在本教學中，您會傳送 Amazon SNS 訊息做為模型進入或結束特定狀態時的動作。

監視裝置或程序

如果您要監視多個裝置或程序，請在每個輸入中指定一個欄位，以識別輸入來源的特定裝置或處理程序。(請參閱中的`key`欄位`CreateDetectorModel`。) 當一個新的設備被識別（在由標識的輸入字段中看到一個新的值`key`），一個檢測器被創建。（每個檢測器都是檢測器模型的實例。）然後，新的檢測器繼續響應來自該設備的輸入，直到其檢測器型號被更新或刪除。

如果您正在監視單個進程（即使有多個設備或子進程正在發送輸入），則不會指定唯一的標識`key`字段。在這種情況下，當第一個輸入到達時，會創建一個檢測器（實例）。

將消息作為輸入發送到檢測器模型

有幾種方法可以將消息作為輸入從設備或進程發送到AWS IoT Events檢測器中，而不需要您對消息執行其他格式化。在本教學課程中，您會使用AWS IoT主控台為將AWS IoT Events郵件資料轉寄至的AWS IoT Core規則引擎撰寫AWS IoT Events動作規則。要做到這一點，您可以通過名稱識別輸入。然後，您繼續使用AWS IoT控制台來生成一些作為輸入轉發到的消息AWS IoT Events。

您如何知道檢測器模型中需要哪些狀態？

若要判斷您的偵測器模型應具有哪些狀態，請先決定您可以採取哪些動作。例如，如果您的汽車使用汽油運行，您可以在開始旅行時查看燃料表，以查看是否需要加油。在這裡，你有一個動作：告訴司機「去加油」。您的探測器型號需要兩種狀態：「汽車不需要燃料」和「汽車確實需要燃料」。一般而言，您想要為每個可能的動作定義一個狀態，並在不需要動作時再定義一個狀態。即使操作本身更複雜，這也可以工作。例如，您可能想查找並包含有關在哪裡可以找到最近的加油站或最便宜的價格的信息，但是當您將消息發送到「去獲取加油」時，請執行此操作。

若要決定接下來要進入的狀態，請查看輸入。輸入包含決定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可協助您做出決定的欄位。在此範例中，您需要一個輸入來告訴您目前的燃油液位（「百分比滿」）。也許您的車向您發送了幾條不同的消息，每個消息都有幾個不同的字段。若要建立此輸入，您必須選取訊息和報告目前氣體計量表液位的欄位。您將要採取的行程長度（「到目的地的距離」）可以硬編碼以簡化操作；您可以使用平均行程長度。您將根據輸入進行一些計算（該百分比完全轉換為多少加侖？ 考慮到您擁有的加侖和平均「每加侖英里數」，平均行程長度大於您可以旅行的里數）。您可以執行這些計算，並在事件中傳送訊息。

到目前為止，你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，並決定是否進入第二個狀態。這是一個過渡事件。 (transitionEvents位於州的onInput事件清單中。 接收處於此第一個狀態的輸入時，事件會執行轉換至第二個狀態 (如果事condition件符合)。當您到達第二個狀態時，只要您進入狀態，就會立即傳送訊息。（您使用的onEnter事件。 在進入第二個狀態，這個事件發送消息。 無需等待另一個輸入到達。）還有其他類型的事件，但這就是你需要一個簡單的例子。

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

你錯過了什麼嗎？ 是的，你如何回到第一個「汽車不需要燃料」狀態？ 加滿油箱後，輸入會顯示一個完整的油箱。在第二種狀態下，您需要一個轉換事件回到收到輸入時發生的第一個狀態（在第二個狀態的onInput:事件中）。它應該過渡回到第一個狀態，如果它的計算顯示你現在有足夠的氣體讓你到達你想去的地方。

這是基礎知識。一些檢測器模型通過添加反映重要輸入的狀態而變得更加複雜，而不僅僅是可能的操作。例如，偵測器模型中可能有三種狀態來追蹤溫度：「正常」狀態、「過熱」狀態和「潛在問題」狀態。當溫度升高到一定水平以上但還沒有變得太熱時，您會轉換到潛在的問題狀態。除非警報保持在此溫度超過 15 分鐘，否則您不想發送警報。如果在此之前溫度恢復正常，則檢測器會轉換回正常狀態。如果計時器過期，檢測器轉換到過熱狀態並發送警報，只是為了小心。你可以使用變量和一組更複雜的事件條件做同樣的事情。但實際上，使用其他狀態來存儲計算結果通常更容易。

你怎麼知道你是否需要一個檢測器或幾個實例？

若要決定您需要多少個執行個體，請問自己：「您有興趣瞭解哪些執行個體？」 假設您想知道今天的天氣。是下雨（狀態）？ 你需要拿一把雨傘（行動）嗎？ 您可以擁有一個報告溫度的傳感器，另一個報告濕度的傳感器以及報告氣壓，風速和方向以及降水的其他傳感器。但是，您必須一起監視所有這些傳感器，以確定天氣狀態（下雨，雪，陰天，陽光明媚）以及要採取的適當措施（拿起雨傘或塗抹防曬霜）。儘管感測器數量有多少，您仍希望一個偵測器執行個體監控天氣狀態，並通知您要採取哪些動作。

但是，如果您是您所在地區的天氣預報員，則可能會有多個此類傳感器陣列的實例，位於整個地區的不同位置。每個位置的人都需要知道該位置的天氣情況。在這種情況下，您需要檢測器的多個實例。每個位置的每個傳感器報告的數據必須包含您指定為該key字段的字段。此欄位可讓您AWS IoT Events為該區域建立偵測器執行個體，然後在偵測器執行個體繼續到達時繼續將此資訊路由至該偵測器執行個體。不再有毀掉的頭髮或曬傷的鼻子!

本質上，如果您有一種情況（一個進程或一個位置）要監視，則需要一個檢測器實例。如果您有許多情況（位置，進程）要監視，則需要多個檢測器實例。

簡單的 step-by-step 例子

在此範例中，我們使用AWS CLI命令呼叫 AWS IoT Events API 來建立一個偵測器，該偵測器會為引擎的兩種狀態建立模型：正常狀態和過壓條件。

當引擎中測量的壓力超過特定閾值時，模型會轉換到過壓狀態，並傳送 Amazon Simple Notification Service (Amazon SNS) 訊息，以警示技術人員此情況。當壓力降至連續三個壓力讀數的閾值以下時，模型會返回正常狀態並傳送另一則 Amazon SNS 訊息，以確認情況已清除。我們需要低於壓力閾值的連續三個讀數，以消除在非線性恢復階段或一次性異常恢復讀數時可能出現過壓/正常消息的口吃。

以下是建立偵測器的步驟概觀。

創建輸入。

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。方法是將訊息作為輸入傳送到 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，但要求您的裝置或程序能夠透過 AWS IoT Events SDK 或AWS CLI.
- 在中AWS IoT Core，為將AWS IoT Events訊息資料轉寄至的AWS IoT Core規則引擎撰寫AWS IoT Events動作規則。這通過名稱標識輸入。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。這種方法通常需要較少的設備的計算能力。
- 在中AWS IoT Analytics，使用 CreateDataset作業建立具有指定AWS IoT Events輸入`contentDeliveryRules`的資料集，其中資料集內容會自動傳送。如果您想要根據中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，裝置才能以這種方式傳送資料。若要這麼做，請為每個輸入指定一個名稱，並指定輸入監控內送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，定義評估傳入輸入以偵測重要事件的條件式 (Boolean) 邏輯。偵測到事件時，它可以變更狀態，或使用其他AWS服務啟動自訂或預先定義的動作。您可以定義其他事件，這些事件會在進入或退出狀態時啟動動作，以及在符合條件時 (選擇性)。

監視多個設備或進程

如果您正在監視多個設備或進程，並且想要單獨跟踪它們中的每個設備或進程，請在每個輸入中指定一個字段，用於標識輸入來自的特定設備或進程。請參閱中的`key`欄位`CreateDetectorModel`。當識別出新裝置時 (在由指定的輸入欄位中看到新值`key`)，就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，直到其偵測器型號更新或刪除為止。您擁有盡可能多的唯一檢測器（實例），因為輸入`key`字段中有唯一值。

監控單一裝置或程序

如果您正在監視單個進程（即使有多個設備或子進程正在發送輸入），則不會指定唯一的標識`key`字段。在這種情況下，當第一個輸入到達時，會創建一個檢測器（實例）。例如，您可能在房屋的每個房間都有溫度感應器，但只有一個 HVAC 裝置來加熱或冷卻整個房屋。因此，即使每個房間的乘客都希望他們的投票（輸入）佔上風，您也只能將其控制為單個過程。

從您的設備或流程發送消息作為輸入到檢測器模型

我們描述了從設備或進程作為輸入到AWS IoT Events檢測器的輸入發送消息的幾種方法。建立輸入並建立偵測器模型之後，就可以開始傳送資料了。

> **Note**
>
> 當您建立偵測器模型或更新現有的偵測器模型時，新的或更新的偵測器模型需要幾分鐘的時間才會開始接收訊息並建立偵測器 (執行個體)。如果檢測器模型已更新，在此期間，您可能會繼續看到以前版本的行為。

主題

創建輸入以捕獲設備數據

舉例來說，假設您的裝置以下列格式傳送訊息。

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

您可以使用以下AWS CLI命令創建一個輸入來捕獲pressure數據和motorid（標識發送消息的特定設備）。

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

該文件pressureInput.json包含以下內容。

```
{
  "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，請記得先從您的裝置或程序收集以 JSON 檔案的形式收集範例訊息。您可以使用它們從控制台或 CLI 創建輸入。

建立偵測器模型以表示裝置狀態

在中創建輸入以捕獲設備數據，您input根據報告來自馬達壓力資料的訊息建立了一個。為了繼續這個例子，這裡是一個檢測器模型，該檢測器模型可以響應電動機中的過壓事件。

您建立兩個狀態：Normal"" 和 "Dangerous"。每個偵測器 (實例) 在建立時都會進入 Normal "" 狀態。當具有 key "" 唯一值的輸入到達時，會建立執行個motorid體。

如果偵測器執行個體收到 70 以上的壓力讀數，則會進入 Dangerous "" 狀態並傳送 Amazon SNS 訊息做為警告。如果連續三個輸入的壓力讀數恢復正常 (小於 70)，偵測器會返回「Normal」狀態，並傳送另一則 Amazon SNS 訊息，並全部清晰傳送。

此範例偵測器模型假設您已建立兩個 Amazon SNS 主題，其 Amazon 資源名稱 (ARN) 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 簡單通知服務開發人員指南，更具體地說，Amazon 簡單通知服務 API 參考中的CreateTopic操作文件。

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

您可以使用以下AWS CLI命令創建檢測器模型。

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

該文件"motorDetectorModel.json"包含以下內容。

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

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

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

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

將消息作為輸入發送到檢測器

您現在已經定義了一個輸入，用於識別從設備發送的消息中的重要字段（請參閱創建輸入以捕獲設備數據）。在上一節中，您建立了對馬達中detector model的過壓事件做出回應 (請參閱建立偵測器模型以表示裝置狀態)。

要完成此示例，請從設備（在本例中為AWS CLI已安裝的計算機）發送消息作為檢測器的輸入。

> **Note**
>
> 當您創建檢測器模型或更新現有的檢測器模型時，新的或更新的檢測器模型開始接收消息並創建檢測器（實例）需要幾分鐘的時間。如果您更新偵測器模型，在此期間，您可能會繼續看到以前版本的行為。

使用以下AWS CLI命令發送包含超出閾值的數據的消息。

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

您必須變更每封傳送的訊息messageId中的。如果您不對其進行更改，則AWS IoT Events系統會刪除重複的消息。 AWS IoT Events如果郵件與過去五分鐘內傳送messageID的其他郵件相同，則會忽略該郵件。

此時，將創建一個檢測器（實例）來監視電動機的事件"Fulton-A32"。該檢測器在創建時進入"Normal"狀態。但是因為我們發送了一個高於閾值的壓力值，所以它立即過渡到"Dangerous"狀態。如此一來，偵測器會將訊息傳送至其 ARN 所在的 Amazon SNS 端點。arn:aws:sns:us-east-1:123456789012:underPressureAction

執行下列AWS CLI命令，以傳送包含低於壓力閾值的資料的訊息。

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

該文件normalPressureMessage.json包含以下內容。

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

每次messageId在五分鐘內呼叫BatchPutMessage指令時，您都必須變更檔案中的。再傳送兩次訊息。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"並重新進入"Normal"狀態。

> **Note**
>
> 您可以使用一次發送多個消息BatchPutMessage。不過，這些訊息的處理順序並不保證。為確保訊息 (輸入) 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。

以下是本節描述的偵測器模型範例所建立的 SNS 訊息承載範例。

事件「壓力閾值突破」

```
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「正常壓力恢復」

```
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，其目前狀態也會顯示在 SNS 訊息承載中。

訊息承載包含傳送訊息時 (也就是執行 SNS 動作時) 偵測器 (執行個體) 狀態的相關資訊。您可以使用該https://docs.aws.amazon.com/iotevents/latest/apireference/API_iotevents-data_DescribeDetector.html操作來獲取有關檢測器狀態的類似信息。

偵測器型號限制和限制

建立偵測器模型時，需要考慮下列事項。

如何使用**actions**欄位

該actions字段是對象的列表。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

在其他情況下transitionEvents，這condition是必要的，且是選用的。

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

條件運算式評估的結果應為布林值。如果結果不是 Boolean 值，則相當於false且不會啟動actions或轉換至事件中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的，則其新值將無法使用，或用於評估相同群組中其他事件中的條件。在相同或欄位中的事件條件中，無法使用新值onInput，onEnter或無法使用此新onExit值。

在檢測器模型定義中設置evaluationMethod參數以更改此行為。當設定evaluationMethod為時SERIAL，會更新變數，並以事件的定義順序評估事件條件。否則，當設定evaluationMethod為BATCH或預設為它時，狀態中的變數會更新，而且只有在評估所有事件條件之後，才會執行狀態中的事件。

在"Dangerous"狀態下，在現onInput場中，"$variable.pressureThresholdBreached"在滿足條件時遞減 1（當電流輸入的壓力小於或等於 70 時）。"Pressure Okay"

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

```
            }
          ]
        }
```

當"$variable.pressureThresholdBreached"達到 0 時，檢測器應該轉換回到"Normal"狀態（即，當檢測器收到三個連續的壓力讀數小於或等於 70 時）。中的"BackToNormal"事件transitionEvents必須測試小"$variable.pressureThresholdBreached"於或等於 1 (非 0)，並再次驗證由指定的目前值"$input.PressureInput.sensorData.pressure"是否小於或等於 70。

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

否則，如果條件僅測試變數的值，則兩個正常讀數後跟過壓讀數將滿足條件並轉換回狀"Normal"態。條件正在查看前一次處理輸入期間給出的值。"$variable.pressureThresholdBreached"在"Overpressurized"事件中，變數的值會重設為 3，但請記住，這個新值尚未提供給任何人使用condition。

預設情況下，每次控制項進入onInput欄位時，在中指定的任何動作變更之前，只condition能看到變數的值，就像處理輸入開始時一樣onInput。onEnter和的情況也是如此onExit。當我們進入或退出狀態時對變量所做的任何更改都不適用於相同onEnter或onExit字段中指定的其他條件。

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，並使用新模型來重新建立偵測器。在新的檢測器模型生效並且新輸入到達之後，它們將被重新創建。在此期間，先前版本的偵測器模型所產生的偵測器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型號所定義的警示。

輸入鍵中的空格

輸入鍵中允許使用空格，但是對索引鍵的參照必須以反引號括住，無論是在 input 屬性的定義中，還是在運算式中參照索引鍵的值。例如，給定如下所示的消息有效負載：

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

使用以下內容定義輸入。

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

在條件表達式中，您也必須使用反引號引用任何此類鍵的值。

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

一個評論的例子：暖通空調溫度控制

以下一些示例 JSON 文件具有內聯註釋，這使得它們無效的 JSON。這些範例的完整版本 (不含註解) 可在下列位置取得暖通空調溫度控制。

背景介紹

這個範例會實作恆溫器控制模型，讓您能夠執行下列動作。

- 只定義一個可用於監視和控制多個區域的檢測器模型。每個區域都會建立偵測器執行個體。
- 從每個控制區域的多個感測器擷取溫度資料。
- 變更區域的溫度設定點。
- 設定每個區域的操作參數，並在例證使用中時重設這些參數。

- 從某個區域動態新增或刪除感測器。
- 指定保護加熱和冷卻裝置的最小執行時間。
- 拒絕異常傳感器讀數。
- 定義緊急設定點，如果任何一個傳感器報告的溫度高於或低於給定閾值，則立即進行加熱或冷卻。
- 報告異常讀數和溫度峰值。

輸入定義

我們希望創建一個檢測器模型，我們可以用它來監視和控制幾個不同區域的溫度。每個區域可以有幾個傳感器來報告溫度。我們假設每個區域由一個加熱單元和一個冷卻裝置提供服務，可以打開或關閉以控制該區域中的溫度。每個區域由一個偵測器執行個體控制。

由於我們監視和控制的不同區域可能具有不同的特性，需'seedTemperatureInput'要不同的控制參數，所以我們定義為每個區域提供這些參數。當我們發送這些輸入消息之一時AWS IoT Events，將創建一個新的檢測器模型實例，該實例具有我們要在該區域使用的參數。以下是該輸入的定義。

CLI 指令：

```
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

- 為任何消息中'areaId'接收的每個唯一創建一個新的檢測器實例。請參閱'areaDetectorModel'定義中的'key'欄位。
- 平均溫度可以從該'desiredTemperature'區域激活加熱或冷卻單元'allowedError'之前的變化。
- 如果任何傳感器報告高於溫度'rangeHigh'，則檢測器報告一個尖峰並立即啟動冷卻裝置。
- 如果任何傳感器報告低於溫度'rangeLow'，則檢測器報告一個尖峰並立即啟動加熱單元。
- 如果任何傳感器報告的溫度高於'anomalousHigh'或低於'anomalousLow'，則檢測器會報告異常的傳感器讀數，但忽略報告的溫度讀數。
- 'sensorCount'告訴檢測器有多少傳感器正在報告該區域。檢測器通過為其接收的每個溫度讀數賦予適當的重量係數來計算該區域的平均溫度。因此，檢測器不必跟踪每個傳感器報告的內容，並且可以根據需要動態更改傳感器的數量。但是，如果單個傳感器脫機，則檢測器將無法知道這一點或為其提供津貼。我們建議您建立另一個偵測器型號，專門用於監視每個感測器的連線狀態。具有兩個互補的檢測器模型簡化了兩者的設計。
- 'noDelay'值可以是true或false。加熱或冷卻裝置開啟後，應保持一定的最短時間，以保護裝置的完整性並延長其使用壽命。如果設定'noDelay'為false，偵測器實體會在關閉冷卻和加熱裝置之前強制執行延遲，以確保它們在最短時間內執行。因為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經硬編碼延遲秒數。

用'temperatureInput'於將感測器資料傳輸到偵測器執行個體。

CLI 指令：

```
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

- 示例檢測器實例'sensorId'不使用該實例來直接控制或監視傳感器。它會自動傳遞至偵測器執行個體傳送的通知。從那裡，它可以用來識別失敗的傳感器（例如，定期發送異常讀數的傳感器可能即將失敗），或者已脫機（當它用作監視設備心跳的其他檢測器模型的輸入時）。如果該區域的讀數經常與平均值不同，則還'sensorId'可以幫助識別該區域的溫暖或冷區。
- 用'areaId'於將感測器的資料路由至適當的偵測器執行個體。為任何消息中'areaId'接收的每個唯一創建檢測器實例。請參閱'areaDetectorModel'定義中的'key'欄位。

檢測器型號定義

這個'areaDetectorModel'例子有註釋內聯。

CLI 指令：

```
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"
    }
}
```

BatchUpdateDetector例子

您可以使用此BatchUpdateDetector作業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作業會重設受溫度監控和控制之區域的操作參數。此操作使您無需刪除，重新創建或更新檢測器模型即可執行此操作。

CLI 指令：

```
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": []
}
```

BatchPutMessage例子

Example 1

使用該BatchPutMessage操作發送一"seedTemperatureInput"條消息，該消息在溫度控制和監控下為給定區域設置操作參數。接收到的任何具AWS IoT Events有新的消息都"areaId"會導致創建新的檢測器實例。但是新的偵測器執行個體不會將狀態變更為"idle"並開始監控溫度並控制加熱或冷卻裝置，直到收到新區域的"seedTemperatureInput"訊息為止。

CLI 指令：

```
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

使用該BatchPutMessage操作發送"temperatureInput"消息以報告給定控制和監測區域中傳感器的溫度傳感器數據。

CLI 指令：

```
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

使用此BatchPutMessage作業傳送"seedTemperatureInput"訊息，以變更指定區域中所需溫度的值。

CLI 指令：

```
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": []
}
```

範例：擷取 MQTT 訊息

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訊息代理程式，您可以建立AWS IoT Core主題規則，將郵件資料重新導向至輸入AWS IoT Events。以下是AWS IoT Events主題規則的定義，該規則從 MQTT 主題中取得"areaId"和"sensorId"輸入欄位，以及訊息承載"sensorData.temperature"欄位中的"temp"欄位，並將此資料擷取到我們的。AWS IoT Events "temperatureInput"

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訊息代理程式，您可以建立AWS IoT Core主題規則，將郵件資料重新導向至輸入AWS IoT Events。以下是AWS IoT Events主題規則的定義，該規則從 MQTT 主題中

取得"areaId"和"sensorId"輸入欄位，以及訊息承載"sensorData.temperature"欄位中的"temp"欄位，並將此資料擷取到我們的。AWS IoT Events "temperatureInput"

CLI 指令：

```
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]

如果傳感器發送有關具有以下有效負載"update/temperature/Area51/03"的主題的消息。

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

這會導致數據被攝入，就好AWS IoT Events像已經進行了以下 "BatchPutMessage" API 調用一樣。

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

範例：產生的 Amazon SNS 訊息

以下是"Area51"偵測器執行個體所產生之 SNS 訊息的範例。

```
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"}
```

範例： DescribeDetectorAPI

您可以使用此DescribeDetector作業查看偵測器實體的目前狀態、變數值和計時器。

CLI 指令：

```
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"
    }
}
```

AWS IoT Core規則引擎範例

下列規則會將 AWS IoT Core MQTT 郵件重新發佈為陰影更新要求訊息。我們假設AWS IoT Core事物是針對由檢測器模型控制的每個區域的加熱單元和冷卻單元定義的。在這個例子中，我們已經定義的東西命名"Area51HeatingUnit"和"Area51CoolingUnit"。

CLI 指令：

```
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]

CLI 指令：

```
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]

CLI 指令：

```
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]

CLI 指令：

```
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]

支援的動作

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。

> **Note**
>
> 當您在偵測器模型中定義動作時，您可以針對字串資料類型的參數使用運算式。如需詳細資訊，請參閱表達式。

AWS IoT Events 支援下列可讓您使用計時器或設定變數的動作：

- `setTimer`創建一個計時器。
- `resetTimer`以重設計時器。
- `clearTimer`刪除計時器。
- `setVariable`以建立變數。

AWS IoT Events 支援下列可讓您使用 AWS 服務的動作：

- `iotTopicPublish`以發佈有關 MQTT 主題的訊息。
- `iotEvents`將資料 AWS IoT Events 作為輸入值傳送至。
- `iotSiteWise` 將資料傳送至 AWS IoT SiteWise 中的資產屬性。
- `dynamoDB`將資料傳送至亞 Amazon DynamoDB 資料表。
- `dynamoDBv2`將資料傳送至亞 Amazon DynamoDB 資料表。
- `firehose`將數據發送到 Amazon 數據 Firehose 流。
- `lambda`調用一個 AWS Lambda 函數。
- `sns`以推送通知的形式傳送資料。
- `sqs`將資料傳送到 Amazon SQS 佇列。

使用內建動作

AWS IoT Events 支援下列可讓您使用計時器或設定變數的動作：

- `setTimer`創建一個計時器。
- `resetTimer`以重設計時器。
- `clearTimer`刪除計時器。
- `setVariable`以建立變數。

設定計時器動作

Set timer action

此`setTimer`動作可讓您建立以秒為單位持續時間的計時器。

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`

(選擇性) 計時器的持續時間，以秒為單位。

持續時間運算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，則持續時間運算式的評估結果為 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

此`resetTimer`動作可讓您將計時器設定為之前評估的持續時間運算式結果。

More information (1)

重設計時器時，必須指定下列參數。

`timerName`

計時器的名稱。

AWS IoT Events 重設計時器時，不會重新評估持續時間運算式。

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

此動`clearTimer`作可讓您刪除現有計時器。

More information (1)

刪除計時器時，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

此`setVariable`動作可讓您建立具有指定值的變數。

More information (2)

建立變數時，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。

與其他 AWS 服務合作

AWS IoT Events 支援下列可讓您使用 AWS 服務的動作：

- `iotTopicPublish`以發佈有關 MQTT 主題的訊息。
- `iotEvents`將資料 AWS IoT Events 作為輸入值傳送至。
- `iotSiteWise` 將資料傳送至 AWS IoT SiteWise 中的資產屬性。
- `dynamoDB`將資料傳送至亞 Amazon DynamoDB 資料表。
- `dynamoDBv2`將資料傳送至亞 Amazon DynamoDB 資料表。
- `firehose`將數據發送到 Amazon 數據 Firehose 流。
- `lambda`調用一個 AWS Lambda 函數。
- `sns`以推送通知的形式傳送資料。
- `sqs`將資料傳送到 Amazon SQS 佇列。

> **Important**
>
> - 您必須選擇相同的 AWS 區域，才能使用以 AWS IoT Events 及要使用的 AWS 服務。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。
> - 當您為動作建立其他 AWS 資源時，必須使用相同的「 AWS 區域 AWS IoT Events 」。如果您切換 AWS 區域，存取 AWS 資源可能會發生問題。

根據預設， AWS IoT Events 會針對任何動作以 JSON 格式產生標準承載。此操作有效負載包含所有屬性-值對，這些屬性-值對具有有關檢測器模型實例和觸發操作的事件的信息。若要配置動作承載，您可以使用內容運算式。如需詳細資訊，請參閱 AWS IoT Events API 參考中的表達式和承載資料類型。

AWS IoT Core

IoT topic publish action

此 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，請參閱開發人AWS IoT 員指南 AWS IoT中的訊息代理程式。

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。

您可以使用在偵測器模型中建立的變數或輸入值，在執行階段動態定義 MQTT 主題名稱。

`payload`

（可選）默認有效負載包含所有屬性-值對，這些屬性值對具有有關檢測器模型實例的信息，並且事件觸發了操作。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**
>
> 請確定附加至 AWS IoT Events 服務角色的原則授與`iot:Publish`權限。如需詳細資訊，請參閱 適用於 AWS IoT Events 的 Identity and Access Management。

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

AWS IoT Events

IoT Events action

此 AWS IoT Events 動作可讓您將資料作 AWS IoT Events 為輸入傳送至。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叢集是否發生故障或操作變更，並在此類事件發生時觸發動作。如需詳細資訊，請參閱什麼是 AWS IoT Events？ 在AWS IoT Events 開發人員指南中。

More information (2)

當您將資料傳送至時 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

（可選）默認有效負載包含所有屬性-值對，這些屬性值對具有有關檢測器模型實例的信息，並且事件觸發了操作。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**
>
> 請確定附加至 AWS IoT Events 服務角色的原則授與`iotevents:BatchPutMessage`權限。如需詳細資訊，請參閱 適用於 AWS IoT Events 的 Identity and Access Management。

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 是一項託管服務，可讓您大規模地收集、整理和分析工業設備中的資料。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)

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

> **Important**
>
> 若要接收資料，您必須在中使用現有的資產性質 AWS IoT SiteWise。
>
> - 如果使用主 AWS IoT Events 控台，則必須指定`propertyAlias`以識別目標資產性質。
> - 如果使用 AWS CLI，則必須指定其中一個`propertyAlias`或兩者`assetId`，`propertyId`以識別目標資產性質。

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

(選擇性) 資產屬性的別名。您也可以指定表達式。

assetId

(選擇性) 具有指定屬性的資產 ID。您也可以指定表達式。

propertyId

(選擇性) 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

包含有關屬性值之詳細資訊的結構。

quality

(選擇性) 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

(選擇性) 包含時間戳記資訊的結構。如果未指定此值，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

(選擇性) 從中`timeInSeconds`轉換的納秒偏移量。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`

(選擇性) 資產性質值是必須為`TRUE`或的布林值`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`

(選擇性) 資產性質值為雙精度值。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`

(選擇性) 資產性質值為整數。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`

(選擇性) 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**
>
> 請確定附加至 AWS IoT Events 服務角色的原則授與`iotsitewise:BatchPutAssetPropertyValue`權限。如需詳細資訊，請參閱 適用於 AWS IoT Events 的 Identity and Access Management。

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

Amazon DynamoDB

DynamoDB action

Amazon DynamoDB 可讓您將資料傳送到動 DynamoDB 資料表。DynamoDB 表的一欄會接收您指定的動作裝載中的所有屬性-值配對。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，請參閱什麼是 DynamoDB？ 在 Amazon DynamoDB 發人員指南中。

More information (10)

當您將資料傳送至 DynamoDB 表格的一個資料行時，必須指定下列參數。

`tableName`

接收資料之 DynamoDB 資料表的名稱。此`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`

雜湊鍵的名稱 (也稱為分割區索引鍵)。此`hashKeyField`值必須符合 DynamoDB 表格的分區索引鍵。您也可以指定表達式。

`hashKeyType`

(選擇性) 雜湊鍵的資料類型。雜湊鍵類型的值必須是`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。使`hashKeyValue`用替代模板。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。此`rangeKeyField`值必須符合 DynamoDB 表格的排序索引鍵。您也可以指定表達式。

`rangeKeyType`

(選擇性) 範圍索引鍵的資料類型。雜湊鍵類型的值必須是`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。使`rangeKeyValue`用替代模板。這些範本會在執行時間提供資料。您也可以指定表達式。

operation

(選擇性) 要執行的作業類型。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 表格中刪除現有項目。

`payloadField`

(選擇性) 接收動作承載的 DynamoDB 資料行名稱。預設名稱為 `payload`。您也可以指定表達式。

`payload`

（可選）默認有效負載包含所有屬性-值對，這些屬性值對具有有關檢測器模型實例的信息，並且事件觸發了操作。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，則`DynamoDBAction`會將非 JSON 資料作為二進位資料傳送至 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**
>
> 請確定附加至 AWS IoT Events 服務角色的原則授與`dynamodb:PutItem`權限。如需詳細資訊，請參閱 適用於 AWS IoT Events 的 Identity and Access Management。

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

Amazon DynamoDB (v2)

DynamoDBv2 action

Amazon DynamoDB 作可讓您將資料寫入動 DynamoDB 料表。DynamoDB 表的個別欄會在您指定的動作裝載中接收一個屬性-值配對。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，請參閱什麼是 DynamoDB？ 在 Amazon DynamoDB 發人員指南中。

More information (2)

當您將資料傳送至 DynamoDB 表格的多個資料行時，必須指定下列參數。

`tableName`

接收資料之 DynamoDB 資料表的名稱。您也可以指定表達式。

`payload`

（可選）默認有效負載包含所有屬性-值對，這些屬性值對具有有關檢測器模型實例的信息，並且事件觸發了操作。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**
>
> 裝載類型必須是 JSON。您也可以指定表達式。

> **Note**
>
> 請確定附加至 AWS IoT Events 服務角色的原則授與`dynamodb:PutItem`權限。如需詳細資訊，請參閱 適用於 AWS IoT Events 的 Identity and Access Management。

如需詳細資訊，請參閱 API 參考資料中的動作。AWS IoT Events

Amazon 數據 Firehose

Firehose action

Amazon 資料防火管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 資料 Firehose 端點和配額。Amazon Web Services 一般參考

Amazon 資料 Firehose 是一項全受管服務，可將即時串流資料交付到 Amazon 簡單儲存服務 (Amazon 簡單儲存服務)、Amazon Redshift、Amazon OpenSearch 服務 (服OpenSearch 務) 和 Splunk 等目的地。如需詳細資訊，請參閱什麼是 Amazon 資料 Firehose？ 在 Amazon 數據 Firehose 開發人員指南中。

More information (3)

當您將資料傳送至 Firehose 傳送串流時，您必須指定下列參數。

`deliveryStreamName`

接收資料的 Firehose 交付串流的名稱。

`separator`

(選擇性) 您可以使用字元分隔符號來分隔傳送至 Firehose 傳送串流的連續資料。分隔符號值必須是 `'\n'` (換行符號)、`'\t'` (Tab)、`'\r\n'` (Windows 新行) 或 `','` (逗號)。

`payload`

（可選）默認有效負載包含所有屬性-值對，這些屬性值對具有有關檢測器模型實例的信息，並且事件觸發了操作。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**
>
> 請確定附加至 AWS IoT Events 服務角色的原則授與`firehose:PutRecord`權限。如需詳細資訊，請參閱 適用於 AWS IoT Events 的 Identity and Access Management。

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

AWS Lambda

Lambda action

此動 AWS Lambda 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，請參閱什麼是 AWS Lambda？ 在AWS Lambda 開發人員指南中。

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`

（可選）默認有效負載包含所有屬性-值對，這些屬性值對具有有關檢測器模型實例的信息，並且事件觸發了操作。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**
>
> 請確定附加至 AWS IoT Events 服務角色的原則授與`lambda:InvokeFunction`權限。如需詳細資訊，請參閱 適用於 AWS IoT Events 的 Identity and Access Management。

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。如需支援區域的清單，請參閱中的 Amazon 簡單通知服務端點和配額Amazon Web Services 一般參考。

Amazon 簡單通知服務 (Amazon 簡單通知服務) 是一種 Web 服務，可協調和管理訊息傳送或傳送至訂閱端點或用戶端。如需詳細資訊，請參閱什麼是 Amazon SNS？ 在 Amazon 簡單通知服務開發人員指南中。

> **Note**
>
> Amazon SNS 主題發佈動作不支援 Amazon SNS FIFO (先進先出) 主題。由於規則引擎是完全分散的服務，因此在啟動 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`

（可選）默認有效負載包含所有屬性-值對，這些屬性值對具有有關檢測器模型實例的信息，並且事件觸發了操作。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**
>
> 請確定附加至 AWS IoT Events 服務角色的原則授與`sns:Publish`權限。如需詳細資訊，請參閱 適用於 AWS IoT Events 的 Identity and Access Management。

如需詳細資訊，請參閱 AWS IoT Events API 參考資料TopicPublishAction中的 SNS。

Amazon Simple Queue Service

SQS action

Amazon SQS 動作可讓您將資料傳送到 Amazon SQS 佇列。如需支援區域的清單，請參閱中的 Amazon 簡單佇列服務端點和配額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 簡單佇列服務開發人員指南中的什麼是 Amazon 簡單佇列服務 >。

> **Note**
>
> Amazon SQS 動作不支持 Amazon SQS FIFO（先進先出）主題。由於規則引擎是完全分散的服務，因此在啟動 Amazon SQS 動作時，訊息可能不會以指定的順序顯示。

More information (3)

將資料傳送到 Amazon SQS 佇列時，必須指定下列參數。

`queueUrl`

接收資料之 Amazon SQS 佇列的網址。

`useBase64`

(選擇性) 如果您指定，請將資料 AWS IoT Events 編碼為 Base64 文字。TRUE預設值為 FALSE。

`payload`

（可選）默認有效負載包含所有屬性-值對，這些屬性值對具有有關檢測器模型實例的信息，並且事件觸發了操作。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**
>
> 請確定附加至 AWS IoT Events 服務角色的原則授與`sqs:SendMessage`權限。如需詳細資訊，請參閱 適用於 AWS IoT Events 的 Identity and Access Management。

如需詳細資訊，請參閱 AWS IoT Events API 參考資料TopicPublishAction中的 SNS。

您也可以使用 Amazon SNS 和 AWS IoT Core 規則引擎來觸發 AWS Lambda 函數。這樣就可以使用其他服務 (例如 Amazon Connect)，甚至是公司企業資源規劃 (ERP) 應用程式來採取行動。

> **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 AWS 服務，例如 Amazon Kinesis。從那裡，您可以完成初始分析，然後將結果 AWS IoT Events 作為輸入發送到檢測器。

表達式

AWS IoT Events在建立和更新偵測器模型時，提供了數種指定值的方法。您可以使用運算式來指定常值，或AWS IoT Events在指定特定值之前評估運算式。

語法

您可以在運算式中使用常值、運AWS IoT Events算子、函數、參照和替代範本。

文字

- 整數
- Decimal (小數)
- 字串
- Boolean

電信業者

一元

- 不 (布林值)：**!**
- 不（按位）：**~**
- 減號（算術）：**-**

字串

- 串聯：**+**

  兩個操作數必須是字符串。字符串文字必須用單引號 (**'**) 括起來。

  例如：`'my' + 'string'` -> `'mystring'`

算術

- 加法：**+**

  兩個操作數必須是數字。

- 減法：**-**
- 分部:**/**

除法的結果是一個四捨五入的整數值，除非至少有一個運算元 (除數或紅利) 是十進位值。

- 乘法：*****

按位（整數）

- 或者：**|**

  例如：13 | 5-> 13

- 和：**&**

  例如：13 & 5-> 5

- 異或：**^**

  例如：13 ^ 5-> 8

- 不：**~**

  例如：~13-> -14

Boolean

- 小於：**<**
- 小於或等於：**<=**
- 等於：**==**
- 不等於：**!=**
- 大於或等於：**>=**
- 大於：**>**
- 和：**&&**
- 或者：**||**

> **Note**
>
> 當的子表達式||包含未定義的數據時，該子表達式被視為。`false`

括號

您可以使用括號將運算式中的詞彙分組。

函數

內建函數

timeout("*timer-name*")

評估指定的計時器是true否已經過去。將「計時###」取代為您定義的計時器名稱，並用引號加上引號。在事件動作中，您可以定義計時器，然後啟動計時器、重設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。

處於一種狀態的計時器集可以在不同的狀態下引用。在輸入參照計時器的狀態之前，您必須先造訪建立計時器的狀態。

例如，一個檢測器模型有兩種狀態， TemperatureChecked 和 RecordUpdated。您在 TemperatureChecked 狀態中建立了計時器。您必須先訪問該 TemperatureChecked 州，然後才能在 RecordUpdated 狀態下使用計時器。

為了確保準確性，應設置計時器的最短時間為 60 秒。

> **Note**
>
> timeout()true僅返回在實際計時器到期之後的第一次檢查，false然後返回。

convert(*type*, *expression*)

評估轉換為指定類型的運算式值。類#值必須是StringBoolean、或Decimal。使用下列其中一個關鍵字或評估為包含關鍵字之字串的運算式。只有下列轉換成功並傳回有效值：

- 布爾-> 字符串

  返回字符串"true"或"false"。
- 十進制-> 字符串
- 字符串-> 布爾
- 字符串-> 小數

  指定的字串必須是十進位數字的有效表示，否convert()則會失敗。

如果convert()沒有返回有效值，則它所屬的表達式也是無效的。此結果相當於false且不會觸發actions或轉移到nextState指定為運算式發生的事件的一部分。

isNull(*expression*)

評估表達式是true否返回 null。例如，如果輸入MyInput收到訊息{ "a": null }，則下列評估結果為true，但isUndefined($input.MyInput.a)評估為false。

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

isUndefined(*expression*)

評估表示式是true否未定義。例如，如果輸入MyInput收到訊息{ "a": null }，則下列評估結果為false，但isNull($input.MyInput.a)評估為true。

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

triggerType("*type*")

類#值可以是"Message"或"Timer"。true如下列範例所示，評估計時器是否已過期，因此正在評估其顯示的事件條件。

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

或者收到輸入消息。

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

currentInput("*input*")

評估是true否因為已接收到指定的輸入訊息而評估其顯示的事件條件。例如，如果輸入Command收到訊息{ "value": "Abort" }，則下列評估結果為true。

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

使用此函數可驗證是否正在評估條件，因為已接收到特定輸入且計時器尚未過期，如下列運算式所示。

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

字符串匹配函數

startsWith(*expression1*, *expression2*)

評估第一個字串運算式是true否以第二個字串運算式開頭。例如，如果輸入MyInput收到訊息{ "status": "offline"}，則下列評估結果為true。

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

兩個運算式都必須評估為字串值。如果任一運算式未評估為字串值，則函數的結果未定義。不會執行任何轉換。

endsWith(*expression1*, *expression2*)

評估第一個字串運算式是true否以第二個字串運算式結尾。例如，如果輸入MyInput收到訊息{ "status": "offline" }，則下列評估結果為true。

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

兩個運算式都必須評估為字串值。如果任一運算式未評估為字串值，則函數的結果未定義。不會執行任何轉換。

contains(*expression1*, *expression2*)

評估第一個字串運算式是true否包含第二個字串運算式。例如，如果輸入MyInput收到訊息{ "status": "offline" }，則下列評估結果為true。

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

兩個運算式都必須評估為字串值。如果任一運算式未評估為字串值，則函數的結果未定義。不會執行任何轉換。

按位整數操作函數

bitor(*expression1*, *expression2*)

評估整數運算式的位元 OR (二進位 OR 運算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 }，則下列評估結果為13。

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

兩個運算式都必須評估為整數值。如果任何一個表達式不評估為整數值，則該函數的結果是未定義的。不會執行任何轉換。

bitand(*expression1*, *expression2*)

評估整數運算式的位元 AND (二進位 AND 運算會針對整數的對應位元執行)。例如，如果輸入MyInput收到訊息{ "value1": 13, "value2": 5 }，則下列評估結果為5。

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

兩個運算式都必須評估為整數值。如果任何一個表達式不評估為整數值，則該函數的結果是未定義的。不會執行任何轉換。

bitxor(*expression1*, *expression2*)

評估整數運算式的位元異或 (二進位 XOR 運算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 }，則下列評估結果為8。

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

兩個運算式都必須評估為整數值。如果任何一個表達式不評估為整數值，則該函數的結果是未定義的。不會執行任何轉換。

bitnot(*expression*)

評估整數運算式的位元 NOT (二進位 NOT 運算會針對整數的位元執行)。例如，如果輸入MyInput收到訊息{ "value": 13 }，則下列評估結果為-14。

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

兩個運算式都必須評估為整數值。如果任何一個表達式不評估為整數值，則該函數的結果是未定義的。不會執行任何轉換。

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，如果您有為其定義inputDefinition.attributes.jsonPath項目TemperatureInput的名稱輸入，則這些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
```

對於其值為陣列的欄位，您可以使用來參考陣列的成員[*n*]。例如，給定以下值：

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

該值78.8可以使用以下命令進行引用。

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

Variables

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

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

例如，如果您有一個名TechnicianID為您使用定義的變數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。您無法為運算式中的變數指派值。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 (規則運算式) 模式之識別碼的參考中`[a-zA-Z][a-zA-Z0-9_]*`，您必須將這些識別碼括在 backtick (`) 中。例如，對以名稱為欄位命名之輸入`MyInput`的參照，`_value`必須將此欄位指定為`$input.MyInput.`_value``。

當您在運算式中使用參照時，請檢查下列項目：

- 當您使用具有一或多個運算子的參考做為運算元時，請確定您參考的所有資料型別都相容。

  例如，在下列運算式中，integer 2 是==和&&運算子的運算元。為了確保操作數是兼容的，`$variable.testVariable + 1`並且`$variable.testVariable`必須引用整數或小數。

  此外，整數`1`是運算子的`+`運算元。因此，`$variable.testVariable`必須引用整數或小數。

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

- 當您使用參考作為傳遞給函數的引數時，請確保該函數支持您引用的數據類型。

  例如，下列`timeout("time-name")`函數需要以雙引號做為引數的字串。如果您使用`#####`值的參考，則必須使用雙引號引用字串。

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

  > **Note**
  >
  > 對於`convert(type, expression)`函數，如果您使用`##`值的引用，則引用的評估結果必須是`String`、`Decimal`、或`Boolean`。

AWS IoT Events表示式支援整數、十進位、字串和布林資料類型。下表提供不相容的類型對清單。

不相容的類型對
整數，字符串
整數，布爾

不相容的類型對
十進制，字符串
十進制，布爾
字符串，布爾

替代範本

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

會將字串${}識別為內插字串。expression可以是任何AWS IoT Events運算式。這包括運算子、函數和參照。

例如，您使用動SetVariableAction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"
`'Sensor 10: ${$variable.SensorID == 10}'`	"Sensor 10: true"
`'{\"sensor\":\"${$variable.SensorID + 1}\"}'`	"{\"sensor\":\"11\"}"
`'{\"sensor\":${$variable.SensorID + 1}}'`	"{\"sensor\":11}"

運算式用法

您可以通過以下方式在檢測器模型中指定值：

- 在主控台中輸入支援的運算AWS IoT Events式。
- 將運算式作為參數傳遞至 AWS IoT Events API。

運算式支援常值、運算子、函數、參考和替代範本。

> **Important**
> 您的運算式必須參考整數、十進位、字串或布林值。

撰寫AWS IoT Events運算式

請參閱下列範例，以協助您撰寫AWS IoT Events運算式：

常值

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

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

參考

對於參考，您必須指定變數或輸入值。

- 下列輸入參照十進位數字`10.01`。

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

- 下列變數會參考字串，`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。

- 下列運算式的評估結果為字串、50.018 in Fahrenheit。

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

- 下列運算式的評估結果為字串、{\"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。

- 下列運算式的評估結果為字串、Greenhouse Temperature Table 2000-01-01。

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

偵測器模型範例

本節包含檢測器模型和輸入的示例。

主題

暖通空調溫度控制

背景故事

此範例實作具有下列功能的溫度控制模型 (恆溫器)：

- 您定義的一個偵測器模型，可以監控和控制多個區域。（將為每個區域創建一個檢測器實例。）
- 每個偵測器實例都會從放置在每個控制區域的多個感測器接收溫度資料
- 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作業參數，並隨時變更這些參數。
- 您可以隨時將感測器新增至某個區域或從區域刪除感測器。
- 您可以啟用時間加熱和冷卻裝置的最小運行，以保護它們免受損壞。
- 偵測器將拒絕並報告異常感測器讀數。
- 您可以定義緊急溫度設定點。如果任何一個傳感器報告的溫度高於或低於您定義的設定點，則加熱或冷卻裝置將立即啟動，並且檢測器將報告該溫度峰值。

這個例子演示了以下功能功能：

- 建立事件偵測器模型。
- 創建輸入。
- 將輸入內嵌到偵測器模型中。

- 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。
- 請參閱在條件定時器和根據條件設置定時器.
- 採取傳送 Amazon SNS 和 MQTT 訊息的動作。

輸入定義

A 用"seedTemperatureInput"於創建一個區域的檢測器實例，並定義其操作參數。

使用 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."
    }
}
```

必要時，"temperatureInput"應由每個區域中的每個傳感器發送 A。

使用 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."
    }
}
```

檢測器型號定義

定"areaDetectorModel"義了每個檢測器實例的工作方式。每個"state machine"執行個體都會擷取溫度感測器讀數，然後根據這些讀數變更狀態並傳送控制訊息。

使用 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"
    }
}
```

BatchPutMessage例子

在此範例中，用"BatchPutMessage"於建立區域的偵測器執行個體，並定義初始操作參數。

使用 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，用"BatchPutMessage"於報告區域中單一感測器的溫度感測器讀數。

使用 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，用"BatchPutMessage"於變更區域所需的溫度。

使用 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"
    }
}
```

BatchUpdateDetector 例子

在此範例中，用"BatchUpdateDetector"於變更工作偵測器執行個體的操作參數。

使用 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規則引擎範例

下列規則會將 AWS IoT Events MQTT 郵件重新發佈為陰影更新要求訊息。我們假設AWS IoT Core事物被定義為加熱單元和由檢測器模型控制的每個區域的冷卻單元。

在這個例子中，我們已經定義的東西命名"Area51HeatingUnit"和"Area51CoolingUnit"。

使用 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]

使用 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]

使用 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]

使用 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]

起重機

背景故事

許多起重機的操作員想要檢測機器何時需要維護或更換，並觸發適當的通知。每台起重機都有一個電機。馬達會發出有關壓力和溫度的訊息（輸入）。操作員需要兩個級別的事件檢測器：

- 起重機級事件檢測器
- 馬達級事件偵測器

使用來自馬達的訊息 (包含中繼資料與"craneId"和"motorId")，操作員可以使用適當的路由來執行這兩個層級的事件偵測器。符合事件條件時，應將通知傳送至適當的 Amazon SNS 主題。操作員可以配置檢測器模型，以便不會引發重複的通知。

這個例子演示了以下功能功能：

- 創建，讀取，更新，刪除（CRUD）輸入。
- 創建，讀取，更新，刪除（CRUD）的事件檢測器模型和不同版本的事件檢測器。
- 將一個輸入路由至多個事件偵測器。
- 將輸入攝入到檢測器模型中。
- 評估觸發條件和生命週期事件。
- 能夠在條件中引用狀態變量，並根據條件設置其值。
- 運行時協調與定義, 狀態, 觸發器評估, 和操作執行程序.
- 使用 SNS 目標執ActionsExecutor行中的動作。

命令

```
#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
```

偵測器模型

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"
}
```

要更新現有的檢測器模型。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"
}
```

要更新現有的檢測器模型。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"
}
```

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"}
        ]
    }
}
```

訊息

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\"}"
        }
    ]
}
```

利用感測器和應用偵測事件

該檢測器型號是AWS IoT Events控制台提供的模板之一。為了您的方便，它包含在此處。

```
{
    "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"
    }
}
```

裝置 HeartBeat

該檢測器型號是AWS IoT Events控制台提供的模板之一。為了您的方便，它包含在此處。

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

```
                        {
                            "eventName": "To_normal",
                            "actions": [],
                            "condition":
"currentInput(\"AWS_IoTEvents_Blueprints_Heartbeat_Input\")",
                            "nextState": "Normal"
                        }
                    ],
                    "events": []
                },
                "stateName": "Offline",
                "onEnter": {
                    "events": [
                        {
                            "eventName": "Send_notification",
                            "actions": [
                                {
                                    "sns": {
                                        "targetArn": "sns-topic-arn"
                                    }
                                }
                            ],
                            "condition": "true"
                        }
                    ]
                },
                "onExit": {
                    "events": []
                }
            },
            {
                "onInput": {
                    "transitionEvents": [
                        {
                            "eventName": "Go_offline",
                            "actions": [],
                            "condition": "timeout(\"awake\")",
                            "nextState": "Offline"
                        }
                    ],
                    "events": [
                        {
                            "eventName": "Reset_timer",
                            "actions": [
```

```
                                {
                                    "resetTimer": {
                                        "timerName": "awake"
                                    }
                                }
                            ],
                            "condition":
 "currentInput(\"AWS_IoTEvents_Blueprints_Heartbeat_Input\")"
                        }
                    ]
                },
                "stateName": "Normal",
                "onEnter": {
                    "events": [
                        {
                            "eventName": "Create_timer",
                            "actions": [
                                {
                                    "setTimer": {
                                        "seconds": 300,
                                        "timerName": "awake"
                                    }
                                }
                            ],
                            "condition":
 "$input.AWS_IoTEvents_Blueprints_Heartbeat_Input.value > 0"
                        }
                    ]
                },
                "onExit": {
                    "events": []
                }
            }
        ],
        "initialStateName": "Normal"
    }
}
```

ISA 警報

該檢測器型號是AWS IoT Events控制台提供的模板之一。為了您的方便，它包含在此處。

```
{
```

```
    "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"
}
```

簡單的報警

該檢測器型號是AWS IoT Events控制台提供的模板之一。為了您的方便，它包含在此處。

```
{
    "detectorModelDefinition": {
        "states": [
            {
                "onInput": {
```

```
                    "transitionEvents": [
                        {
                            "eventName": "not_fixed",
                            "actions": [],
                            "condition": "timeout(\"snoozeTime\")",
                            "nextState": "Alarming"
                        },
                        {
                            "eventName": "reset",
                            "actions": [],
                            "condition":
"$input.AWS_IoTEvents_Blueprints_Simple_Alarm_Input.command == \"reset\"",
                            "nextState": "Normal"
                        }
                    ],
                    "events": [
                        {
                            "eventName": "DND",
                            "actions": [
                                {
                                    "setVariable": {
                                        "variableName": "dnd_active",
                                        "value": "1"
                                    }
                                }
                            ],
                            "condition":
"$input.AWS_IoTEvents_Blueprints_Simple_Alarm_Input.command == \"dnd\""
                        }
                    ]
                },
                "stateName": "Snooze",
                "onEnter": {
                    "events": [
                        {
                            "eventName": "Create Timer",
                            "actions": [
                                {
                                    "setTimer": {
                                        "seconds": 120,
                                        "timerName": "snoozeTime"
                                    }
                                }
                            ],
```

```
                            "condition": "true"
                        }
                    ]
                },
                "onExit": {
                    "events": []
                }
            },
            {
                "onInput": {
                    "transitionEvents": [
                        {
                            "eventName": "out_of_range",
                            "actions": [],
                            "condition":
"$input.AWS_IoTEvents_Blueprints_Simple_Alarm_Input.value > $variable.threshold",
                            "nextState": "Alarming"
                        }
                    ],
                    "events": [
                        {
                            "eventName": "Create Config variables",
                            "actions": [
                                {
                                    "setVariable": {
                                        "variableName": "threshold",
                                        "value":
"$input.AWS_IoTEvents_Blueprints_Simple_Alarm_Input.threshold"
                                    }
                                }
                            ],
                            "condition": "$variable.threshold != $variable.threshold"
                        }
                    ]
                },
                "stateName": "Normal",
                "onEnter": {
                    "events": [
                        {
                            "eventName": "Init",
                            "actions": [
                                {
                                    "setVariable": {
                                        "variableName": "dnd_active",
```

```
                                    "value": "0"
                                }
                            }
                        ],
                        "condition": "true"
                    }
                ]
            },
            "onExit": {
                "events": []
            }
        },
        {
            "onInput": {
                "transitionEvents": [
                    {
                        "eventName": "reset",
                        "actions": [],
                        "condition":
 "$input.AWS_IoTEvents_Blueprints_Simple_Alarm_Input.command == \"reset\"",
                        "nextState": "Normal"
                    },
                    {
                        "eventName": "acknowledge",
                        "actions": [],
                        "condition":
 "$input.AWS_IoTEvents_Blueprints_Simple_Alarm_Input.command == \"acknowledge\"",
                        "nextState": "Snooze"
                    }
                ],
                "events": [
                    {
                        "eventName": "Escalated Alarm Notification",
                        "actions": [
                            {
                                "sns": {
                                    "targetArn": "arn:aws:sns:us-
west-2:123456789012:escalatedAlarmNotification"
                                }
                            }
                        ],
                        "condition": "timeout(\"unacknowledgeTIme\")"
                    }
                ]
```

```
                },
                "stateName": "Alarming",
                "onEnter": {
                    "events": [
                        {
                            "eventName": "Alarm Notification",
                            "actions": [
                                {
                                    "sns": {
                                        "targetArn": "arn:aws:sns:us-
west-2:123456789012:alarmNotification"
                                    }
                                },
                                {
                                    "setTimer": {
                                        "seconds": 300,
                                        "timerName": "unacknowledgeTIme"
                                    }
                                }
                            ],
                            "condition": "$variable.dnd_active != 1"
                        }
                    ]
                },
                "onExit": {
                    "events": []
                }
            }
        ],
        "initialStateName": "Normal"
    },
    "detectorModelDescription": "This detector model is used to detect if a monitored
 device is in an Alarming State.",
    "roleArn": "arn:aws:iam::123456789012:role/IoTEventsRole",
    "key": "alarmId"
}
```

使用 警示進行監控

AWS IoT Events警示可協助您監控資料是否有變更。數據可以是您為設備和流程測量的指標。您可以建立警示，在超出閾值時傳送通知。警報可協助您偵測問題、簡化維護作業，並最佳化設備與流程的效能。

警報是警報模型的實例。警報模型指定要偵測的內容、何時傳送通知、收到通知的人員等等。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將從資料衍生的輸入屬性路由至適當的警示。如果您監視的資料超出指定範圍，則會叫用警示。您還可以確認鬧鐘或將其設置為貪睡模式。

使用 AWS IoT SiteWise

您可以使用AWS IoT Events警示來監視中的資產屬性AWS IoT SiteWise。 AWS IoT SiteWise將資產屬性值傳送至AWS IoT Events警示。 AWS IoT Events將警示狀態傳送至AWS IoT SiteWise。

AWS IoT SiteWise還支持外部警報。如果您在以外使用警報，AWS IoT SiteWise且有可傳回警示狀態資料的解決方案，則可以選擇外部警示。外部警報包含擷取警報狀態資料的量測屬性。

AWS IoT SiteWise不會評估外部警報的狀態。此外，當鬧鐘狀態變更時，您無法確認或暫停外部鬧鐘。

您可以使用 SiteWise 監視功能來檢視 Monitor 入口網站中外部警 SiteWise 示的狀態。

如需詳細資訊，請參閱《使用指南》中的AWS IoT SiteWise使用警示監視資料和《監SiteWise 視器應用指南》中的使用警示進行監視。

確認流程

建立警示模型時，您可以選擇是否啟用確認流程。如果您啟用確認流程，您的小組會在警示狀態變更時收到通知。您的團隊可以確認警報並留下便條。例如，您可以包含警示的資訊，以及解決問題所要採取的動作。如果您監視的資料超出指定範圍，則會叫用警示。

警報具有下列狀態：

`DISABLED`

當警示處於狀DISABLED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狀NORMAL態。

NORMAL

當警示處於狀NORMAL態時，就可以評估資料。

ACTIVE

如果警示處於狀ACTIVE態，則會叫用警示。您正在監視的資料超出指定的範圍。

ACKNOWLEDGED

當警示處於狀ACKNOWLEDGED態時，就會叫用警示並確認警示。

LATCHED

警報已叫用，但是您在一段時間後並未確認警報。警示會自動變更為NORMAL狀態。

SNOOZE_DISABLED

當警示處於此狀SNOOZE_DISABLED態時，會在指定的時間段內停用警示。貪睡時間過後，鬧鐘會自動變更為狀態NORMAL。

建立鬧鐘模型

您可以使用AWS IoT Events警報來監控資料，並在超出閾值時收到通知。警報提供您用來建立或設定警報模型的參數。您可以使用AWS IoT Events主控台或 AWS IoT Events API 建立或設定警示模型。設定警示模型時，變更會在新資料送達時生效。

請求

建立鬧鐘模型時，適用下列需求。

- 您可以建立警示模型來監視中的輸入屬性AWS IoT Events或中的資產屬性AWS IoT SiteWise。
  - 如果您選擇監視中的輸入屬性AWS IoT Events，請在建立警報模型之前執行下列動作：
    - 步驟 1：在創建輸入中閱讀概述。
    - 步驟 2：閱讀說明以在導航窗格中創建輸入。
  - 如果您選擇監視資產屬性，則必須在中建立資產模型，然AWS IoT SiteWise後才能建立警示模型。
- 您必須具有可讓警示執行動作和存取AWS資源的 IAM 角色。如需詳細資訊，請參閱設定的權限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區域中。

建立警示模型 (主控台)

以下說明如何建立警示模型以監AWS IoT Events控主控台中的AWS IoT Events屬性。

1. 登入 AWS IoT Events 主控台。
2. 在功能窗格中，選擇 [警報模型]。
3. 在 [警報模型] 頁面上，選擇 [建立鬧鐘模型]。
4. 在 [警示型號詳細資料] 區段中，執行下列動作：

   a. 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**
   >
   > 如果選擇AWS IoT SiteWise資產性質，則必須在中建立資產模型AWS IoT SiteWise。

   a. 選擇AWS IoT Events輸入屬性。

   b. 選擇輸入。

   c. 選擇輸入屬性鍵。此輸入屬性用作建立警示的索引鍵。 AWS IoT Events將與此鍵相關聯的輸入路由到警報。

      > **Important**
      >
      > 如果輸入訊息承載不包含此輸入屬性金鑰，或者金鑰不在金鑰中指定的相同 JSON 路徑中，則訊息將無法擷取。AWS IoT Events

6. 在 [臨界值定義] 區段中，您可以定義AWS IoT Events用來變更警示狀態的輸入屬性、臨界值和比較運算子。

   a. 針對輸入屬性，選擇您要監視的屬性。

      每次此輸入屬性接收到新資料時，都會對其進行評估以確定警示的狀態。

   b. 在「運算子」中，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。

您可以從以下選項中進行選擇：

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

c. 對於臨界值，請輸入數字或在AWS IoT Events輸入中選擇屬性。 AWS IoT Events將此值與您選擇的輸入屬性值進行比較。

d. (選擇性) 對於嚴重性，請使用團隊瞭解的數字來反映此警示的嚴重性。

7. (選擇性) 在 [通知設定] 區段中，設定警報的通知設定。

   您最多可以新增 10 則通知。對於「通知 1」，請執行下列操作：

   a. 在「通訊協定」中，從下列選項中選擇：

      - 電子郵件和簡訊-警示會傳送 SMS 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 SMS 通知。

   b. 針對寄件者，指定可傳送有關此警示通知的電子郵件地址。

      若要將更多電郵地址新增至寄件者清單，請選擇 [新增寄件者]。

   c. (選擇性) 針對「收件者」，選擇收件者。

      若要將更多使用者新增至收件者清單，請選擇 [新增使用者]。您必須先將新使用者新增至 IAM 身分中心商店，才能將使用者新增至警示模型。如需詳細資訊，請參閱管理收件者。

   d. (選擇性) 針對其他自訂訊息，輸入描述警示偵測到的內容以及收件者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 [啟用]，警示狀態變更時會收到通知。您必須在警報狀態恢復正常之前確認通知。

- 如果您選擇「停用」，則不需要採取任何動作。當量測回到指定範圍時，警報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「權限」，選擇下列其中一個選項：

- 您可以從AWS政策範本建立新角色，並AWS IoT Events自動為您建立 IAM 角色。
- 您可以使用現有的 IAM 角色，該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，您可以編輯管理鬧鐘通知的AWS Lambda功能。為您的AWS Lambda函數選擇下列其中一個選項：

- 創建一個新AWS Lambda函數-為您AWS IoT Events創建一個新AWS Lambda函數。
- 使用現有AWS Lambda函數-透過選擇AWS Lambda函數名稱來使用現有AWS Lambda函數。

如需可能動作的詳細資訊，請參閱與其他 AWS 服務合作。

d. (選擇性) 對於「設定狀態」動作，您可以新增警示狀態變更時要AWS IoT Events採取的一或多個動作。

10. （可選）您可以添加標籤來管理您的警報。如需詳細資訊，請參閱標記您的 AWS IoT Events 資源。
11. 選擇 建立 。

響應警報

如果啟用了確認流程，則會在警示狀態變更時收到通知。若要回應鬧鐘，您可以確認、停用、啟用、重設或暫停鬧鐘。

回應警示 (主控台)

以下說明如何回應AWS IoT Events主控台中的警示。

1. 登入 AWS IoT Events 主控台。
2. 在功能窗格中，選擇 [警報模型]。

3. 選擇目標警報模型。
4. 在警報清單區段中，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-警示會變更為狀ACKNOWLEDGED態。
   - 停用-警示會變更為狀DISABLED態。
   - 啟用-警示會變更為狀NORMAL態。
   - 重設-警示會變更為狀NORMAL態。
   - 暫停，然後執行下列動作：
     1. 選擇延遲時間長度或輸入自訂延遲時間長度。
     2. 選擇 儲存 。

     警報變更為狀SNOOZE_DISABLED態

   如需警示狀態的詳細資訊，請參閱確認流程。

回應警示 (API)

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

- BatchAcknowledgeAlarm
- BatchDisableAlarm
- BatchEnableAlarm
- BatchResetAlarm
- BatchSnoozeAlarm

管理警報通知

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也可以建立新函數。

建立 Lambda 函數

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。

請求

當您建立警示的 Lambda 函數時，適用下列需求：

- 如果您的警示傳送電子郵件或簡訊通知，您必須擁有可AWS Lambda與 Amazon SES 和 Amazon SNS 搭配使用的 IAM 角色。

  範例政策：

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ses:GetIdentityVerificationAttributes",
                "ses:SendEmail",
                "ses:VerifyEmailIdentity"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "sns:Publish",
                "sns:OptInPhoneNumber",
                "sns:CheckIfPhoneNumberIsOptedOut"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Deny",
            "Action": [
                "sns:Publish"
            ],
            "Resource": "arn:aws:sns:*:*:*"
        }
    ]
}
```

- 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AWS IoT EventsAmazon Web Services 一般參考. AWS Lambda

部署一個 Lambda 函數

本教學課程使用AWS CloudFormation範本來部署 Lambda 函數。此範本會自動建立 IAM 角色，讓 Lambda 函數與 Amazon SES 和 Amazon SNS 搭配使用。

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

1. 在設備的終端中，運行`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 使用者指南》中的安裝 AWS CLI。
2. 運行`aws configure list`以檢查您是否在AWS區域AWS CLI中配置了具有本教程的所有AWS資源。如需詳細資訊，請參閱《使用指南》AWS CLI中的AWS Command Line Interface〈配置〉
3. 下載 CloudFormation 模板，通知。。模板。

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

4. 解壓縮內容並以 `notificationLambda.template.yaml` 儲存在本機。
5. 開啟裝置上的終端機，然後瀏覽至下載`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 範本以自訂 Lambda 函數及其行為。

> **Note**
>
> AWS Lambda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《使用指南》中的〈AWS CloudFormation使用堆疊〉。

建立自訂 Lambda 函數

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，需符合下列需求。

- 新增允許 Lambda 函數執行指定動作和存取AWS資源的權限。
- 如果您使用由提供的 Lambda 函數AWS IoT Events，請務必選擇 Python 3.7 執行階段。

示例 Lambda 函數：

```
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？章節。

CloudFormation 範本

使用下列 CloudFormation 範本建立您的 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"
                Resource: "*"
              - Effect: "Deny"
                Action:
                  - "sns:Publish"
                Resource: "arn:aws:sns:*:*:*"
  NotificationLambdaFunction:
    Type: AWS::Lambda::Function
    Properties:
      Role: !GetAtt NotificationLambdaRole.Arn
      Runtime: python3.7
      Handler: index.lambda_handler
      Timeout: 300
      MemorySize: 3008
      Code:
        ZipFile: |
          import boto3
          import json
          import logging
          import datetime
          logger = logging.getLogger()
          logger.setLevel(logging.INFO)
```

```
          ses = boto3.client('ses')
          sns = boto3.client('sns')
          def check_value(target):
            if target:
              return True
            return False

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

          # Check whether the phone holder has opted out of receiving SMS messages from
your account
          def check_phone_number(phone_number):
            try:
              result = sns.check_if_phone_number_is_opted_out(phoneNumber=phone_number)
              if (result['isOptedOut']):
                  logger.info('phoneNumber {} is not opt in of receiving SMS messages.
Phone number must be opt in first.'.format(phone_number))
                  return False
              return True
            except Exception as e:
              logging.error('Your phone number {} must be in E.164 format in SSO.
Exception thrown: {}'.format(phone_number, e))
              return False

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

          def lambda_handler(event, context):
```

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

            emails = event.get('emailConfigurations', [])
            logger.info('Start Sending Emails')
            for email in emails:
              from_adr = email.get('from')
              to_adrs = email.get('to', [])
              cc_adrs = email.get('cc', [])
              bcc_adrs = email.get('bcc', [])
              msg = default_msg + '\n' + email.get('additionalMessage', '')
              subject = email.get('subject', alarm_msg)
              fa_ver = check_email(from_adr)
              tas_ver = check_emails(to_adrs)
              ccas_ver = check_emails(cc_adrs)
              bccas_ver = check_emails(bcc_adrs)
              if (fa_ver and tas_ver and ccas_ver and bccas_ver):
                ses.send_email(Source=from_adr,
                               Destination={'ToAddresses': to_adrs, 'CcAddresses':
 cc_adrs, 'BccAddresses': bcc_adrs},
                               Message={'Subject': {'Data': subject}, 'Body': {'Text':
 {'Data': msg}}})
                logger.info('Emails have been sent')

            logger.info('Start Sending SNS message to SMS')
            sns_configs = event.get('smsConfigurations', [])
            for sns_config in sns_configs:
              sns_msg = default_msg + '\n' + sns_config.get('additionalMessage', '')
              phone_numbers = sns_config.get('phoneNumbers', [])
```

```
            sender_id = sns_config.get('senderId')
            for phone_number in phone_numbers:
                if check_phone_number(phone_number):
                  if check_value(sender_id):
                    sns.publish(PhoneNumber=phone_number, Message=sns_msg,
MessageAttributes={'AWS.SNS.SMS.SenderID':{'DataType': 'String','StringValue':
sender_id}})
                  else:
                    sns.publish(PhoneNumber=phone_number, Message=sns_msg)
                  logger.info('SNS messages have been sent')
```

使用由提供的 Lambda 函數 AWS IoT Events

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

- 您必須在 Amazon 簡易電子郵件服務 (Amazon SES) 中驗證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 簡易電子郵件服務開發人員指南中的 Amazon SES 驗證電子郵件地址。

  如果您收到驗證連結，請按一下連結以驗證您的電子郵件地址。您也可以檢查垃圾郵件文件夾中的驗證電子郵件。
- 如果您的鬧鐘傳送簡訊通知，電話號碼必須使用 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：

國家/地區	本地電話號碼	E.164 格式化的數字
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/地區電話代碼，請前往國家代碼 .org。

  Lambda 函數AWS IoT Events會檢查您是否使用 E.164 格式的電話號碼。但是，它不會驗證電話號碼。如果您確定輸入的電話號碼正確無誤，但沒有收到簡訊通知，您可能會聯絡電信業者。運營商可能會阻止消息。

管理收件者

AWS IoT Events使用 AWS IAM Identity Center (IAM 身分中心) 來管理警示收件者的 SSO 存取。若要啟用警示以傳送通知給收件者，您必須啟用 IAM 身分中心，並將收件者新增至您的 IAM 身分中心商店。如需詳細資訊，請參閱使用指南中的新增使用AWS IAM Identity Center者。

Important

- 您必須為AWS IoT Events、AWS Lambda和 IAM 身分中心選擇相同的AWS區域。
- AWSOrganizations 一次僅支援一個 IAM 身分中心區域。如果您想要讓 IAM 身分中心在不同的區域可用，您必須先刪除目前的 IAM 身分中心組態。如需詳細資訊，請參閱AWS IAM Identity Center使用者指南中的 IAM 身分中心區域資料。

AWS IoT Events 中的安全性

雲端安全是 AWS 最重視的一環。身為 AWS 客戶的您，將能從資料中心和網路架構的建置中獲益，以滿足組織最為敏感的安全要求。

安全是 AWS 與您共同的責任。共同的責任模型 將此描述為雲端 本身 的安全和雲端內部的安全：

- 雲端本身的安全：AWS 負責保護在 AWS Cloud 中執行 AWS 服務的基礎設施。AWS 也提供您可安全使用的服務。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。如要了解適用於 AWS IoT Events 的合規計劃，請參閱合規計劃範圍內的 AWS 服務。
- 雲端內部的安全：您的責任取決於所使用的 AWS 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。

本文件會協助您了解使用 AWS IoT Events 時共同責任模型的適用情形。下列主題說明如何將 AWS IoT Events 設定為達到您的安全及合規目標。您還將學習如何使用其他可以幫助您監控和保護AWS IoT Events資源的AWS服務。

主題

適用於 AWS IoT Events 的 Identity and Access Management

AWS Identity and Access Management (IAM) 是一種 AWS 服務，讓管理員能夠安全地控制對 AWS 資源的存取權限。IAM 管理員會控制誰可經身分身分驗證 (已登入) 和授權 (具有許可) 來使用 AWS IoT Events 資源。IAM 是一種您可以免費使用的 AWS 服務。

主題

- 進一步了解
- AWS IoT Events 搭配 IAM 的運作方式
- AWS IoT Events 身分型政策範例
- 預防跨服務混淆代理人
- 對 AWS IoT Events 身分與存取進行疑難排解

物件

AWS Identity and Access Management (IAM) 的使用方式會不同，需視您在 AWS IoT Events 中所執行的工作而定。

服務使用者：如果使用 AWS IoT Events 執行任務，管理員會為您提供所需的憑證和許可。隨著您為了執行作業而使用的 AWS IoT Events 功能數量變多，您可能會需要額外的許可。了解存取的管理方式可協助您向管理員請求正確的許可。若您無法存取 AWS IoT Events 中的某項功能，請參閱 對 AWS IoT Events 身分與存取進行疑難排解。

服務管理員：如果您負責公司內的 AWS IoT Events 資源，您可能具備 AWS IoT Events 的完整存取權限。您的任務是判斷服務使用者應存取的 AWS IoT Events 功能及資源。接著，您必須將請求提交給您的 IAM 管理員，來變更您服務使用者的許可。檢閱此頁面上的資訊，了解 IAM 的基本概念。若要進一步了解貴公司可搭配 AWS IoT Events 使用 IAM 的方式，請參閱 AWS IoT Events 搭配 IAM 的運作方式。

IAM 管理員：如果您是 IAM 管理員，建議您掌握如何撰寫政策以管理 AWS IoT Events 存取權的詳細資訊。若要檢視您可以在 IAM 中使用的範例 AWS IoT Events 身分型政策，請參閱 AWS IoT Events 身分型政策範例。

使用身分驗證

身分驗證是使用身分憑證登入 AWS 的方式。您必須以 AWS 帳戶根使用者、IAM 使用者身分，或擔任 IAM 角色進行 驗證 (登入至 AWS)。

您可以使用透過身分來源 AWS IAM Identity Center 提供的憑證，以聯合身分登入 AWS。(IAM Identity Center) 使用者、貴公司的單一登入身分驗證和您的 Google 或 Facebook 憑證都是聯合身分的範例。您以聯合身分登入時，您的管理員先前已設定使用 IAM 角色的聯合身分。您 AWS 藉由使用聯合進行存取時，您會間接擔任角色。

根據您的使用者類型，您可以登入 AWS Management Console 或 AWS 存取入口網站。如需有關登入至 AWS 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。

如果您是以程式設計的方式存取 AWS，AWS 提供軟體開發套件 (SDK) 和命令列介面 (CLI)，以便使用您的憑證透過密碼編譯方式簽署您的請求。如果您不使用 AWS 工具，您必須自行簽署請求。如需使用建議的方法自行簽署請求的詳細資訊，請參閱《IAM 使用者指南》中的簽署 AWS API 請求。

無論您使用何種身分驗證方法，您可能都需要提供額外的安全性資訊。例如，AWS 建議您使用多重要素驗證 (MFA) 以提高帳戶的安全。如需更多資訊，請參閱《AWS IAM Identity Center 使用者指南》中的多重要素驗證和《IAM 使用者指南》中的在 AWS 中使用多重要素驗證 (MFA)。

AWS 帳戶 根使用者

如果是建立 AWS 帳戶，您會先有一個登入身分，可以完整存取帳戶中所有 AWS 服務 與資源。此身分稱為 AWS 帳戶 根使用者，使用建立帳戶時所使用的電子郵件地址和密碼即可登入並存取。強烈建議您不要以根使用者處理日常作業。保護您的根使用者憑證，並將其用來執行只能由根使用者執行的任務。如需這些任務的完整清單，了解需以根使用者登入的任務，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 使用者是您 AWS 帳戶 中的一種身分，具備單一人員或應用程式的特定許可。建議您盡可能依賴暫時憑證，而不是擁有建立長期憑證 (例如密碼和存取金鑰) 的 IAM 使用者。但是如果特定使用案例需要擁有長期憑證的 IAM 使用者，建議您輪換存取金鑰。如需詳細資訊，請參閱《IAM 使用者指南》https://docs.aws.amazon.com/IAM/latest/UserGuide/best-practices.html#rotate-credentials中的為需要長期憑證的使用案例定期輪換存取金鑰。

IAM 群組是一種指定 IAM 使用者集合的身分。您無法以群組身分登入。您可以使用群組來一次為多名使用者指定許可。群組可讓管理大量使用者許可的過程變得更為容易。例如，您可以擁有一個名為 IAMAdmins 的群組，並給予該群組管理 IAM 資源的許可。

使用者與角色不同。使用者只會與單一人員或應用程式建立關聯，但角色的目的是在由任何需要它的人員取得。使用者擁有永久的長期憑證，但角色僅提供暫時憑證。如需進一步了解，請參閱《IAM 使用者指南》中的建立 IAM 使用者 (而非角色) 的時機。

IAM 角色

IAM 角色是您 AWS 帳戶 中的一種身分，具備特定許可。它類似 IAM 使用者，但不與特定的人員相關聯。您可以在 AWS Management Console 中透過切換角色來暫時取得 IAM 角色。您可以透過呼叫 AWS CLI 或 AWS API 操作，或是使用自訂 URL 來取得角色。如需使用角色的方法詳細資訊，請參閱《IAM 使用者指南》中的使用 IAM 角色。

使用暫時憑證的 IAM 角色在下列情況中非常有用：

- 聯合身分使用者存取 — 如需向聯合身分指派許可，請建立角色，並為角色定義許可。當聯合身分進行身分驗證時，該身分會與角色建立關聯，並取得由角色定義的許可。如需有關聯合角色的詳細資訊，請參閱《IAM 使用者指南》https://docs.aws.amazon.com/IAM/latest/UserGuide/id_roles_create_for-idp.html中的為第三方身分供應商建立角色。如果您使用 IAM Identity Center，則需要設定許可集。為控制身分驗證後可以存取的內容，IAM Identity Center 將許可集與 IAM 中的角色相關聯。如需有關許可集的資訊，請參閱《AWS IAM Identity Center 使用者指南》中的許可集。
- 暫時 IAM 使用者許可 – IAM 使用者或角色可以擔任 IAM 角色來暫時針對特定任務採用不同的許可。
- 跨帳户存取權 – 您可以使用 IAM 角色，允許不同帳戶中的某人 (信任的委託人) 存取您帳戶中的資源。角色是授予跨帳户存取權的主要方式。但是，針對某些 AWS 服務，您可以將政策直接連接到資源 (而非使用角色作為代理)。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 角色與資源類型政策的差異。
- 跨服務存取 – 有些 AWS 服務 會使用其他 AWS 服務 中的功能。例如，當您在服務中進行呼叫時，該服務通常會在 Amazon EC2 中執行應用程式或將物件儲存在 Amazon Simple Storage Service (Amazon S3) 中。服務可能會使用呼叫主體的許可、使用服務角色或使用服務連結角色來執行此作業。
  - 轉發存取工作階段 (FAS)：當您使用 IAM 使用者或角色在 AWS 中執行動作時，系統會將您視為主體。當您使用某些服務時，您可能會執行一個動作，而該動作之後會在不同的服務中啟動另一個動作。FAS 使用主體的許可呼叫 AWS 服務，搭配請求 AWS 服務 以向下游服務發出請求。只有在服務收到需要與其他 AWS 服務 或資源互動才能完成的請求之後，才會提出 FAS 請求。在此情況下，您必須具有執行這兩個動作的許可。如需提出 FAS 請求時的政策詳細資訊，請參閱《轉發存取工作階段》。
  - 服務角色：服務角色是服務擔任的 IAM 角色，可代表您執行動作。IAM 管理員可以從 IAM 內建立、修改和刪除服務角色。如需詳細資訊，請參閱《IAM 使用者指南》中的建立角色以委派許可給 AWS 服務 服務。
  - 服務連結角色 – 服務連結角色是一種連結到 AWS 服務 的服務角色類型。服務可以擔任代表您執行動作的角色。服務連結角色會顯示在您的 AWS 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。
- 在 Amazon EC2 上執行的應用程式 – 針對在 EC2 執行個體上執行並提出 AWS CLI 和 AWS API 請求的應用程式，您可以使用 IAM 角色來管理暫時憑證。這是在 EC2 執行個體內儲存存取金鑰的較好方式。如需指派 AWS 角色給 EC2 執行個體並提供其所有應用程式使用，您可以建立連接到執行個體的執行個體設定檔。執行個體設定檔包含該角色，並且可讓 EC2 執行個體上執行的程式取得暫時憑證。如需詳細資訊，請參閱《IAM 使用者指南》中的利用 IAM 角色來授予許可給 Amazon EC2 執行個體上執行的應用程式。

如需了解是否要使用 IAM 角色或 IAM 使用者，請參閱《IAM 使用者指南》中的建立 IAM 角色 (而非使用者) 的時機。

使用政策管理存取權

您可以透過建立政策並將其附加到 AWS 身分或資源，在 AWS 中控制存取。政策是 AWS 中的一個物件，當其和身分或資源建立關聯時，便可定義其許可。AWS 會在主體 (使用者、根使用者或角色工作階段) 發出請求時評估這些政策。政策中的許可，決定是否允許或拒絕請求。大部分政策以 JSON 文件形式儲存在 AWS 中。如需 JSON 政策文件結構和內容的詳細資訊，請參閱《IAM 使用者指南》中的 JSON 政策概觀。

管理員可以使用 AWS JSON 政策來指定誰可以存取哪些內容。也就是說，哪個主體在什麼條件下可以對什麼資源執行哪些動作。

預設情況下，使用者和角色沒有許可。若要授與使用者對其所需資源執行動作的許可，IAM 管理員可以建立 IAM 政策。然後，管理員可以將 IAM 政策新增至角色，使用者便能擔任這些角色。

IAM 政策定義該動作的許可，無論您使用何種方法來執行操作。例如，假設您有一個允許 `iam:GetRole` 動作的政策。具備該政策的使用者便可以從 AWS Management Console、AWS CLI 或 AWS API 取得角色資訊。

身分型政策

身分型政策是可以附加到身分 (例如 IAM 使用者、使用者群組或角色) 的 JSON 許可政策文件。這些政策可控制身分在何種條件下能對哪些資源執行哪些動作。若要了解如何建立身分類型政策，請參閱《IAM 使用者指南》中的建立 IAM 政策。

身分型政策可進一步分類成內嵌政策或受管政策。內嵌政策會直接內嵌到單一使用者、群組或角色。受管政策則是獨立的政策，您可以將這些政策附加到 AWS 帳戶 中的多個使用者、群組和角色。受管政策包含 AWS 管理政策和客戶管理政策。如需瞭解如何在受管政策及內嵌政策間選擇，請參閱 IAM 使用者指南中的在受管政策和內嵌政策間選擇。

其他政策類型

AWS 支援其他較少見的政策類型。這些政策類型可設定較常見政策類型授與您的最大許可。

- 許可界限 – 許可範圍是一種進階功能，可供您設定身分型政策能授予 IAM 實體 (IAM 使用者或角色) 的最大許可。您可以為實體設定許可界限。所產生的許可會是實體的身分型政策和其許可界限的交集。會在 `Principal` 欄位中指定使用者或角色的資源型政策則不會受到許可界限限制。所有這類政

策中的明確拒絕都會覆寫該允許。如需許可範圍的更多相關資訊，請參閱《IAM 使用者指南》中的 IAM 實體許可範圍。

- 服務控制政策 (SCP) – SCP 是 JSON 政策，可指定 AWS Organizations 中組織或組織單位 (OU) 的最大許可。AWS Organizations 服務可用來分組和集中管理您企業所擁有的多個 AWS 帳戶。若您啟用組織中的所有功能，您可以將服務控制政策 (SCP) 套用到任何或所有帳戶。SCP 會限制成員帳戶中實體的許可，包括每個 AWS 帳戶根使用者。如需組織和 SCP 的更多相關資訊，請參閱《AWS Organizations 使用者指南》中的 SCP 運作方式。
- 工作階段政策 – 工作階段政策是一種進階政策，您可以在透過編寫程式的方式建立角色或聯合使用者的暫時工作階段時，作為參數傳遞。所產生工作階段的許可會是使用者或角色的身分型政策和工作階段政策的交集。許可也可以來自資源型政策。所有這類政策中的明確拒絕都會覆寫該允許。如需更多資訊，請參閱《IAM 使用者指南》中的工作階段政策。

多種政策類型

將多種政策類型套用到請求時，其結果形成的許可會更為複雜、更加難以理解。如需瞭解 AWS 在涉及多種政策類型時如何判斷是否允許一項請求，請參閱 IAM 使用者指南中的政策評估邏輯。

進一步了解

如需 AWS IoT Events 身分和存取權管理的詳細資訊，請繼續參閱以下頁面：

- AWS IoT Events 搭配 IAM 的運作方式
- 對 AWS IoT Events 身分與存取進行疑難排解

AWS IoT Events 搭配 IAM 的運作方式

在您使用 IAM 來管理對 AWS IoT Events 的存取之前，您應該先了解可與 AWS IoT Events 搭配使用的 IAM 功能有哪些。若要取得 AWS IoT Events 和其他 AWS 服務如何搭配 IAM 使用的詳細資訊，請參閱 IAM 使用者指南中的可搭配 IAM 使用的 AWS 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。政策動作的名稱通常會和相關聯的 AWS API 操作相同。政策會使用動作來授予執行相關聯操作的許可。

AWS IoT Events 中的政策動作會在動作之前使用以下字首：`iotevents:`。例如，若要授與某人使用 AWS IoT Events `CreateInput` API 作業建立AWS IoT Events輸入的權限，您可以將該`iotevents:CreateInput`動作包含在他們的政策中。若要授與某人透過 AWS IoT Events `BatchPutMessage` API 作業傳送輸入的權限，您可以將`iotevents-data:BatchPutMessage`動作包含在他們的政策中。政策陳述式必須包含 `Action` 或 `NotAction` 元素。AWS IoT Events 會定義一組自己的動作，來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 動作的清單，請參閱《IAM 使用者指南》中 AWS IoT Events 定義的動作。

資源

`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。

檢AWS IoT Events測器模型資源具有以下 ARN：

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

如需 ARN 格式的詳細資訊，請參閱 Amazon Resource Name (ARN) 和 AWS 服務命名空間。

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 (例如用來建立資源的動作) 無法在特定資源上執行。在這些情況下，您必須使用萬用字元 (*)。

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

許多 AWS IoT Events API 動作都會用到多項資源。例如，CreateDetectorModel參考其條件陳述式中的輸入，因此使用者必須擁有使用輸入和偵測器模型的權限。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 ARN 的清單，請參閱《IAM 使用者指南》中的 AWS IoT Events 定義的資源。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。若您為單一條件索引鍵指定多個值，AWS 會使用邏輯 OR 操作評估條件。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 不提供任何服務專用條件索引鍵，但它支援一些全域條件索引鍵的使用。若要查看所有AWS全域條件金鑰，請參閱 IAM 使用者指南中的AWS全域條件內容金鑰。」

範例

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

AWS IoT Events 資源型政策

AWS IoT Events 不支援資源類型政策。」 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

您可以將標籤連接到 AWS IoT Events 資源，或是在請求中將標籤傳遞給 AWS IoT Events。若要根據標籤控制存取，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。

若要檢視身分型政策範例，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##。

AWS IoT Events IAM 角色

IAM 角色是您 AWS 帳戶 中具備特定許可的實體。

將臨時憑證與 AWS IoT Events 搭配使用

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。您可以呼叫 AWS Security Token Service (AWS STS) API 作業 (例如AssumeRole或) 來取得臨時安全登入資料GetFederationToken。

AWS IoT Events 不支援使用暫時登入資料。

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，使用者和角色不具備建立或修改 AWS IoT Events 資源的權限。他們也無法使用 AWS Management Console、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
- 使用 AWS IoT Events 主控台
- 允許使用者檢視他們自己的許可
- 存取一個AWS IoT Events輸入
- 根據標籤檢視AWS IoT Events輸入

政策最佳實務

身分型政策相當強大。他們可以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 - 若要快速開始使用 AWS IoT Events，請使用 AWS 受管政策來給予員工所需許可。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》中的開始搭配 AWS 受管政策使用許可。
- 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。

- 針對敏感性作業啟用 MFA — 為了提高安全性，使用者必須使用多重要素驗證 (MFA) 來存取敏感資源或 API 作業。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，您必須擁有最低的一組許可。這些許可必須允許您列出和檢視您 AWS 帳戶 中 AWS IoT Events 資源的詳細資訊。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。

為確保那些實體仍可使用 AWS IoT Events 主控台，請也將以下 AWS 受管政策連接到實體。如需詳細資訊，請參閱 IAM 使用者指南中的向使用者新增許可：

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iotevents-data:BatchPutMessage",
                "iotevents-data:BatchUpdateDetector",
                "iotevents:CreateDetectorModel",
                "iotevents:CreateInput",
                "iotevents:DeleteDetectorModel",
                "iotevents:DeleteInput",
                "iotevents-data:DescribeDetector",
                "iotevents:DescribeDetectorModel",
                "iotevents:DescribeInput",
                "iotevents:DescribeLoggingOptions",
                "iotevents:ListDetectorModelVersions",
                "iotevents:ListDetectorModels",
                "iotevents-data:ListDetectors",
                "iotevents:ListInputs",
                "iotevents:ListTagsForResource",
                "iotevents:PutLoggingOptions",
                "iotevents:TagResource",
                "iotevents:UntagResource",
```

```
                "iotevents:UpdateDetectorModel",
                "iotevents:UpdateInput",
                "iotevents:UpdateInputRouting"
            ],
            "Resource": "arn:${Partition}:iotevents:${Region}:${Account}:detectorModel/
${detectorModelName}",
            "Resource": "arn:${Partition}:iotevents:${Region}:${Account}:input/
${inputName}"
        }
    ]
}
```

對於僅呼叫 AWS CLI 或 AWS API 的使用者，您不需要允許其最基本主控台許可。反之，只需允許存取符合您嘗試執行之 API 操作的動作就可以了。

允許使用者檢視他們自己的許可

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。此政策包含在主控台上，或是使用 AWS CLI 或 AWS API 透過編寫程式的方式完成此動作的許可。

```
{
       "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

在此範例中，您想要授與使用者AWS 帳戶存取其中一個AWS IoT Events輸入，`exampleInput`. 您還希望允許用戶添加，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 Amazon 簡單儲存服務 (Amazon S3) 的範例逐步解說，該服務會授予使用者許可並使用主控台進行測試，請參閱範例逐步解說：使用使用者政策控制儲存貯體的存取。

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Sid":"ListInputsInConsole",
         "Effect":"Allow",
         "Action":[
            "iotevents:ListInputs"
         ],
         "Resource":"arn:aws:iotevents:::*"
      },
      {
         "Sid":"ViewSpecificInputInfo",
         "Effect":"Allow",
         "Action":[
            "iotevents:DescribeInput"
         ],
         "Resource":"arn:aws:iotevents:::exampleInput"
      },
```

```
      {
         "Sid":"ManageInputs",
         "Effect":"Allow",
         "Action":[
            "iotevents:CreateInput",
            "iotevents:DeleteInput",
            "iotevents:DescribeInput",
            "iotevents:ListInputs",
            "iotevents:UpdateInput"
         ],
         "Resource":"arn:aws:iotevents:::exampleInput/*"
      }
   ]
}
```

根據標籤檢視AWS IoT Events##

您可以在身分型政策中使用條件，以根據標籤控制存取 AWS IoT Events 資源。此範例顯示如何建立允許檢視##的原則。不過，只有在##標籤Owner具有該使用者名稱的值時，才會授與權限。此政策也會授予在主控台完成此動作的必要許可。

```
{
    "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`#`入，則`#`入必須加上標籤`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。如需更多資訊，請參閱《IAM 使用者指南》中的 IAM JSON 政策元素：條件。

預防跨服務混淆代理人

> **Note**
>
> - 此AWS IoT Events服務僅允許客戶使用角色在建立資源的相同帳號中啟動動作。這意味著不能使用此服務執行混淆的副攻擊。
> - 此頁面可作為客戶查看混淆副問題如何運作的參考，並且在AWS IoT Events服務中允許跨帳戶資源的情況下可以避免此頁面。

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權限的實體執行該動作。在 AWS 中，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。為了預防這種情況，AWS 提供的工具可協助您保護所有服務的資料，而這些服務主體已獲得您帳戶中資源的存取權。

若要限制 AWS IoT Events 為資源提供另一項服務的許可，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。如果 `aws:SourceArn` 值不包含帳戶 ID (例如 Simple Storage Service (Amazon S3) 儲存貯體 ARN)，您必須使用這兩個全域條件內容索引鍵來限制許可。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型號或警示模型。

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:*`。

下列範例說明如何在中使用aws:SourceArn和aws:SourceAccount全域條件前後關聯鍵字AWS IoT Events來避免混淆的副問題。

主題

範例 1：存取偵測器模型

下列角色只能用來存取具 DetectorModel 名foo。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": [
          "iotevents.amazonaws.com"
        ]
      },
      "Action": "sts:AssumeRole",
      "Condition": {
        "StringEquals": {
          "aws:SourceAccount": "account_id"
        },
        "ArnEquals": {
          "aws:SourceArn": "arn:aws:iotevents:region:account_id:detectorModel/foo"
        }
      }
    }
  ]
 }
}
```

範例 2：存取警示模型

以下角色只能用於訪問任何警報模型。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": [
          "iotevents.amazonaws.com"
        ]
      },
      "Action": "sts:AssumeRole",
      "Condition": {
        "StringEquals": {
          "aws:SourceAccount": "account_id"
        },
        "ArnEquals": {
          "aws:SourceArn": "arn:aws:iotevents:region:account_id:alarmModel/*"
        }
      }
    }
  ]
}
```

範例 3：存取指定區域中的資源

下列範例顯示可用來存取指定區域中資源的角色。此範例中的區域為 *us-east-* 1。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": [
          "iotevents.amazonaws.com"
        ]
      },
      "Action": "sts:AssumeRole",
      "Condition": {
        "StringEquals": {
          "aws:SourceAccount": "account_id"
        },
        "ArnEquals": {
```

```
          "aws:SourceArn": "arn:aws:iotevents:us-east-1:account_id:*"
        }
      }
    }
  ]
}
```

範例 4：記錄選項

若要提供記錄選項的角色，您必須允許 IoT Events 中的每個資源都採用此角色。因此，您必須為資源類型和資源名稱使用萬用字元 (*)。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": [
          "iotevents.amazonaws.com"
        ]
      },
      "Action": "sts:AssumeRole",
      "Condition": {
        "StringEquals": {
          "aws:SourceAccount": "account_id"
        },
        "ArnEquals": {
          "aws:SourceArn": "arn:aws:iotevents:region:account_id:*"
        }
      }
    }
  ]
}
```

對 AWS IoT Events 身分與存取進行疑難排解

請使用以下資訊來協助您診斷和修復使用 AWS IoT Events 和 IAM 時發生的常見問題。

主題

- 我未獲授權，不得在 AWS IoT Events 中執行動作

- 我未獲得執行 iam:PassRole 的授權
- 我想要允許 AWS 帳戶 外的人員存取我的 AWS IoT Events 資源

我未獲授權，不得在 AWS IoT Events 中執行動作

若 AWS Management Console 告知您並未獲得執行動作的授權，您必須聯絡您的管理員以取得協助。您的管理員是提供您使用者名稱和密碼的人員。

當 mateojackson IAM 使用者嘗試使用主控台來檢視有關*##*的詳細資料但沒有iotevents:*ListInputs*權限時，就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 允許您傳遞現有的角色至該服務，而無須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 沒有將角色傳遞至該服務的許可。

```
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。

如需任何協助，請聯絡您的 AWS 管理員。您的管理員提供您的登入憑證。

我想要允許 AWS 帳戶 外的人員存取我的 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的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。

如需進一步了解，請參閱以下內容：

- 若要了解 AWS IoT Events 是否支援這些功能，請參閱 AWS IoT Events 搭配 IAM 的運作方式。
- 若要了解如何存取您擁有的所有 AWS 帳戶 所提供的資源，請參閱《IAM 使用者指南》中的將存取權提供給您所擁有的另一個 AWS 帳戶 中的 IAM 使用者。
- 如需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擁有的 AWS 帳戶。
- 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)。
- 若要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 角色與資源型政策的差異。

監控 AWS IoT Events

監控是維護 AWS IoT Events 及您 AWS 解決方案可靠性、可用性和效能的重要部分。您應該從AWS解決方案的所有部分收集監視資料，以便在發生多點失敗時更輕鬆地偵錯。在開始監控 AWS IoT Events 之前，應先建立監控計畫，為下列問題提供解答：

- 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？

下一步是在各個時間點和不同的負載條件下測量效能，以在您的環境中確立 AWS IoT Events 正常效能的基準。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，則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

- 監控工具

- 使用亞馬遜監控 CloudWatch
- 使用 AWS CloudTrail 記錄 AWS IoT Events API 呼叫

監控工具

AWS 提供您可用來監控 AWS IoT Events 的多種工具。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來監看 AWS IoT Events，並在發生錯誤時進行回報：

- Amazon CloudWatch 日誌 — 監控、存放和存取來自AWS CloudTrail或其他來源的日誌檔。如需詳細資訊，請參閱 Amazon CloudWatch 使用者指南中的監控日誌檔。
- Amazon E CloudWatch vents — 匹配事件並將其路由到一個或多個目標函數或串流，以進行變更、擷取狀態資訊並採取糾正措施。有關更多信息，請參閱亞馬遜用 CloudWatch 戶指南中的亞馬遜 CloudWatch 事件是什麼。
- AWS CloudTrail記錄監控 — 在帳戶之間共用記錄檔、即時監控記 CloudTrail 錄檔案，方法是將記錄檔傳送至 CloudWatch 記錄檔、以 Java 撰寫記錄處理應用程式，以及驗證您的記錄檔在傳送之後未變更。 CloudTrail若要取得更多資訊，請參閱《使用指南》中的〈AWS CloudTrail使用 CloudTrail 記錄檔〉。

手動監控工具

監視AWS IoT Events的另一個重要部分是手動監視 CloudWatch 警報未涵蓋的項目。AWS IoT Events CloudWatch、和其他AWS主控台儀表板可提供您AWS環境狀態的 at-a-glance 檢視。建議您也查看 AWS IoT Events 上的日誌檔。

- AWS IoT Events 主控台會顯示：
  - 偵測器模型
  - 偵測器
  - 輸入
  - 設定
- CloudWatch 首頁顯示：
  - 目前警示與狀態

- 警示與資源的圖表
- 服務運作狀態

此外，您可以使用執行 CloudWatch 以下操作：

- 建立自定儀表板來監控您注重的服務
- 繪製指標資料圖表，以對問題進行故障診斷並探索趨勢
- 搜尋與瀏覽您所有的 AWS 資源指標
- 建立與編輯要通知發生問題的警示

使用亞馬遜監控 CloudWatch

當您開發或偵錯偵AWS IoT Events測器模型時，您需要知道正在做什AWS IoT Events麼，以及它遇到的任何錯誤。亞馬遜 CloudWatch 監控您的 Amazon Web Services（AWS）資源和您實時運行AWS的應用程序。有了 CloudWatch，您就可以掌握整個系統的資源使用、應用程式效能和營運狀態。 開發 AWS IoT Events 偵測器模型時啟用 Amazon CloudWatch 記錄包含如何啟用 CloudWatch 記錄的資訊 AWS IoT Events。若要產生如下所示的記錄檔，您必須將詳細程度層級設定為「偵錯」，並提供一或多個偵錯目標，即偵測器型號名稱和選用項目。KeyValue

下列範例顯示由所產生的 CloudWatch DEBUG 層級記錄項目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 AWS CloudTrail 記錄 AWS IoT Events API 呼叫

AWS IoT Events與 (提供中的使用者AWS CloudTrail、角色或服務所採取的動作記錄) 的AWS服務整合AWS IoT Events。 CloudTrail 擷取AWS IoT Events做為事件的所有 API 呼叫，包括來自AWS IoT Events主控台的呼叫，以及從 API 的程AWS IoT Events式碼呼叫。

如果您建立追蹤，您可以啟用持續交付 CloudTrail 事件到 Amazon S3 儲存貯體，包括AWS IoT Events. 如果您未設定追蹤，您仍然可以在 [事件歷程記錄] 中檢視 CloudTrail 主控台中最近的事件。使用收集的資訊 CloudTrail，您可以判斷提出的要求AWS IoT Events、提出要求的 IP 位址、提出要求的人員、提出要求的時間，以及其他詳細資訊。

若要進一步了解 CloudTrail，請參閱使AWS CloudTrail用者指南。

AWS IoT Events中的資訊 CloudTrail

CloudTrail 在您創建AWS帳戶時，您的帳戶已啟用。當活動發生在中時AWS IoT Events，該活動會與事件歷史記錄中的其他AWS服務 CloudTrail 事件一起記錄在事件中。您可以檢視、搜尋和下載 AWS 帳戶的最新事件。如需詳細資訊，請參閱使用 CloudTrail 事件歷程記錄檢視事件。

如需您 AWS 帳戶中正在進行事件的記錄 (包含 AWS IoT Events 的事件)，請建立線索。追蹤可 CloudTrail 將日誌檔交付到 Amazon S3 儲存貯體。根據預設，當您在主控台建立線索時，線索會套用到所有 AWS 區域。該追蹤會記錄來自 AWS 分割區中所有區域的事件，並將日誌檔案交付到您指定的 Amazon S3 儲存貯體。此外，您還可以設定其他AWS服務，以進一步分析 CloudTrail 記錄中收集的事件資料並採取行動。如需詳細資訊，請參閱：

- 建立追蹤的概觀
- CloudTrail 支援的服務與整合
- 設定 Amazon SNS 通知 CloudTrail
- 從多個區域接收 CloudTrail 日誌文件並從多個帳戶接收 CloudTrail 日誌文件

每一筆事件或日誌項目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。
- 該請求是否由另一項 AWS 服務提出。

如需詳細資訊，請參閱CloudTrail 使 userIdentity 元素。 AWS IoT Events動作記錄在 AWS IoT EventsAPI 參考中。

了解 AWS IoT Events 日誌檔案項目

追蹤是一種組態，可讓事件以日誌檔的形式傳遞到您指定的 Amazon S3 儲存貯體。 AWS CloudTrail 記錄檔包含一或多個記錄項目。事件代表來自任何來源的單一請求，包括有關請求的操作，動作的日期和時間，請求參數等信息。 CloudTrail 日誌文件不是公共 API 調用的有序堆棧跟踪，因此它們不會以任何特定順序顯示。

在您的AWS帳戶中啟用日誌記 CloudTrail 錄時，對AWS IoT Events動作進行的大多數 API 調用都會在 CloudTrail 日誌文件中進行跟踪，其中它們與其他AWS服務記錄一起寫入。 CloudTrail 根據時間週期和檔案大小決定何時建立和寫入新檔案。

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。
- 該請求是否由另一項 AWS 服務提出。

您可以視需要將日誌檔存放在 Amazon S3 儲存貯體中，但也可以定義 Amazon S3 生命週期規則以自動存檔或刪除日誌檔。根據預設，您的日誌檔會使用 Amazon S3 伺服器端加密 (SSE) 加密。

若要在日誌檔交付時收到通知，您可以設定 CloudTrail 為在交付新的日誌檔時發佈 Amazon SNS 通知。如需詳細資訊，請參閱設定 CloudTrail 的 Amazon SNS 通知。

您也可以將來自多個 AWS 區域和多個 AWS 帳戶的 AWS IoT Events 日誌檔案，彙總至單一 Amazon S3 儲存貯體。

如需詳細資訊，請參閱從多個區域接 CloudTrail 收 CloudTrail 記錄檔和從多個帳戶接收記錄檔。

下列範例顯示示範`DescribeDetector`動作的 CloudTrail 記錄項目。

```
{
  "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"
}
```

下列範例顯示示範CreateDetectorModel動作的 CloudTrail 記錄項目。

```
{
  "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"
}
```

下列範例顯示示範CreateInput動作的 CloudTrail 記錄項目。

```
{
  "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"
}
```

下列範例顯示示範DeleteDetectorModel動作的 CloudTrail 記錄項目。

```
{
  "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"
}
```

下列範例顯示示範DeleteInput動作的 CloudTrail 記錄項目。

```
{
  "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"
}
```

下列範例顯示示範DescribeDetectorModel動作的 CloudTrail 記錄項目。

```
{
  "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"
}
```

下列範例顯示示範DescribeInput動作的 CloudTrail 記錄項目。

```
{
  "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"
}
```

下列範例顯示示範DescribeLoggingOptions動作的 CloudTrail 記錄項目。

```
{
  "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"
}
```

下列範例顯示示範`ListDetectorModels`動作的 CloudTrail 記錄項目。

```
{
  "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"
}
```

下列範例顯示示範`ListDetectorModelVersions`動作的 CloudTrail 記錄項目。

```
{
  "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"
}
```

下列範例顯示示範`ListDetectors`動作的 CloudTrail 記錄項目。

```
{
  "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"
}
```

下列範例顯示示範`ListInputs`動作的 CloudTrail 記錄項目。

```
{
  "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"
}
```

下列範例顯示示範PutLoggingOptions動作的 CloudTrail 記錄項目。

```
{
  "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"
}
```

下列範例顯示示範UpdateDetectorModel動作的 CloudTrail 記錄項目。

```
{
  "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"
}
```

下列範例顯示示範UpdateInput動作的 CloudTrail 記錄項目。

```
{
  "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"
}
```

AWS IoT Events 的合規驗證

要瞭解 AWS 服務 是否在特定法規遵循方案範圍內，請參閱法規遵循方案範圍內的 AWS 服務，並選擇您感興趣的法規遵循方案。如需一般資訊，請參閱 AWS 法規遵循方案。

您可使用 AWS Artifact 下載第三方稽核報告。如需詳細資訊，請參閱 AWS Artifact 中的下載報告。

您使用 AWS 服務 時的法規遵循責任取決於資料的敏感度、您的公司的合規目標，以及適用的法律和法規。AWS 提供以下資源協助您處理法規遵循事宜：

- 安全與合規快速入門指南 – 這些部署指南討論在 AWS 上部署以安全及合規為重心的基準環境的架構考量和步驟。
- Amazon Web Services 的 HIPAA 安全與法規遵循架構：本白皮書說明公司可如何運用 AWS 來建立符合 HIPAA 規定的應用程式。

> **Note**
>
> 並非全部的 AWS 服務 都符合 HIPAA 資格。如需詳細資訊，請參閱 HIPAA 資格服務參照。

- AWS 合規資源：這組手冊和指南可能適用於您的產業和位置。

- AWS 客戶合規指南：透過合規的角度了解共同的責任模式。這份指南橫跨多個架構 (包含國家標準技術研究所 (NIST)、支付卡產業安全標準委員會 (PCI) 和國際標準組織 (ISO))，總結保護 AWS 服務的最佳實務並將指導方針對應至安全控制。
- AWS Config 開發人員指南中的使用規則評估資源：AWS Config 服務可評估您的資源組態對於內部實務、業界準則和法規的合規狀態。
- AWS Security Hub – 此 AWS 服務 可供您全面檢視 AWS 中的安全狀態。Security Hub 使用安全控制，可評估您的 AWS 資源並檢查您的法規遵循是否符合安全業界標準和最佳實務。如需支援的服務和控制清單，請參閱 Security Hub controls reference。
- AWS Audit Manager – 此 AWS 服務 可協助您持續稽核 AWS 使用情況，以簡化管理風險與法規與業界標準的法規遵循方式。

AWS IoT Events 中的恢復能力

AWS 全球基礎設施是以 AWS 區域與可用區域為中心建置的。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 區域與可用區域的詳細資訊，請參閱 AWS 全球基礎設施。

AWS IoT Events 中的基礎設施安全

作為一種受管服務，AWS IoT Events 受 AWS 全域網路安全的保護。如需有關 AWS 安全服務以及 AWS 如何保護基礎設施的詳細資訊，請參閱 AWS 雲端安全。若要使用基礎設施安全性的最佳實務來設計您的 AWS 環境，請參閱安全性支柱 AWS 架構良好的框架中的基礎設施保護。

您可使用 AWS 發佈的 API 呼叫，透過網路存取。用戶端必須支援下列項目：

- 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。

此外，請求必須使用存取索引鍵 ID 和與 IAM 主體相關聯的私密存取索引鍵來簽署。或者，您可以使用 AWS Security Token Service (AWS STS) 來產生暫時安全憑證來簽署請求。

AWS IoT Events 配額

本AWS 一般參考指南提供帳戶的預設配AWS額。AWS IoT Events除非另有說明，否則每個配額均為每個AWS區域 如需詳細資訊，請參閱AWS 一般參考指南中的AWS IoT Events端點和配額以及AWSService Quotas。

若要要求提高服務配額，請在 Support 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**
>
> - 檢測器模型和輸入的所有名稱在帳戶中必須是唯一的。
> - 您無法在建立偵測器模型和輸入之後變更它們的名稱。

標記您的 AWS IoT Events 資源

為了幫助您管理和組織檢測器模型和輸入，您可以選擇性地以標籤的形式將自己的元數據分配給每個這些資源。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您的 AWS IoT Events 資源，例如依據目的、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，以協助您追蹤依類型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的管理您的資源。

您可以根據新增或套用的標籤搜尋和篩選資源、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〈搭配 IAM 政策使用標籤〉中所述。

為了方便使用，中的標籤編輯器AWS Management Console提供了一種集中、統一的方式來建立和管理標籤。若要取得更多資訊，請參閱〈使用〉中的〈使用標籤編輯器〉AWS Management Console。

您還可使用 AWS CLI 和 AWS IoT Events API 來使用標籤。您可以使用下列指令中的"Tags"欄位，在建立偵測器模型和輸入時，將標籤與檢測器模型和輸入產生關聯：

- 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

您可以編輯標籤金鑰和值，並且可以隨時從資源移除標籤。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。若您將與現有標籤具有相同鍵的標籤新增到該資源，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。

AWS標籤策略中提供了其他資訊。

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 中的 127 個萬國碼字元
- 最大值長度 — UTF-8 中 255 個萬國碼字元
- 標籤鍵與值皆區分大小寫。
- 請勿在標籤名稱或值中使用"aws:"前置詞，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。

使用《使用指南》中的AWS Identity and Access Management標籤控制存取權限有關使用標籤的其他資訊。該指南的 IAM JSON 政策參考部分包含 IAM 中 JSON 政策的元素、變數和評估邏輯的詳細語法、說明和範例。

以下範例政策會套用兩個以標籤為基礎的限制。受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"`)。

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

您也可以將多個標籤值封閉在清單中，以便為指定的標籤鍵指定多個標籤值，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。

AWS IoT Events 疑難排解

請使用這些章節中的資訊進行疑難排解和解決 AWS IoT Events 的問題。

主題

常見AWS IoT Events問題和解決方案

請參閱下一節以疑難排解錯誤，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

當我嘗試創建檢測器模型時出現錯誤。

解決方案

建立偵測器模型時，必須考量下列限制。

- 每個欄位只允許一個動`action`作。

- 這condition是必需的transitionEvents。對於、和OnExit事件來說 OnEnterOnInput，它是可選的。
- 如果condition欄位為空白，則條件運算式的評估結果等於true。
- 條件運算式的評估結果應為布林值。如果結果不是 Boolean 值，則相當於false且不會觸發事件中nextState指定的actions或轉換。

如需詳細資訊，請參閱 偵測器型號限制和限制。

已刪除偵測器模型的更新

我在幾分鐘前更新或刪除了檢測器模型，但仍然通過 MQTT 消息或 SNS 警報從舊檢測器模型獲取狀態更新。

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)，則在刪除所有偵測器執行個體並使用新模型之前會有一段延遲。在此期間，先前版本的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型號所定義的警示。請等待至少七分鐘，然後再重新檢查更新或報告錯誤。

動作觸發失敗 (符合條件時)

符合條件時，偵測器無法觸發動作或轉換至新狀態。

解決方案

驗證檢測器的條件表達式的評估結果是一個布爾值。如果結果不是 Boolean 值，則相當於false且不會觸發事件中nextState指定的action或轉換。如需詳細資訊，請參閱條件運算式語法。

動作觸發失敗 (抽取閾值時)

當條件運算式中的變數達到指定值時，偵測器不會觸發動作或事件轉換。

解決方案

如果您更setVariable新onInput、或 onEnteronExit，則condition在目前處理週期期間評估任何值時，不會使用新值。相反地，會使用原始值，直到目前的循環完成為止。您可以通過在檢測器模型定義中設置evaluationMethod參數來更改此行為。設定evaluationMethod為時SERIAL，會更新變數並以事件的定義順序評估事件條件。當設定evaluationMethod為 BATCH (預設值) 時，變數會更新，而且只有在評估所有事件條件之後才會執行事件。

狀態用法不正確

當我嘗試使用BatchPutMessage.

解決方案

如果您使BatchPutMessage用將多個訊息傳送至輸入，則無法保證處理訊息或輸入的順序。為了保證訂購，請一次發送一個消息，並等待每次BatchPutMessage確認成功。

連線訊息

我嘗試呼叫或叫用 API 時收到('Connection aborted.', error(54, 'Connection reset by peer'))錯誤訊息。

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 發行版或 Windows 7 及更高版本下的默認設置。使用 macOS 可能需要升 OpenSSL。

InvalidRequestException 消息

InvalidRequestException 當我嘗試調UpdateDetectorModel用CreateDetectorModel和 API 時，我得到了。

解決方案

請檢查下列項目以協助解決問題。如需詳細資訊，請參閱CreateDetectorModel及UpdateDetectorModel。

- 請確定您不要同時使用seconds和durationExpression作為SetTimerAction的參數。
- 請確定您的字串運算式durationExpression是有效的。字串運算式可以包含數字、變數 ($variable.<variable-name>) 或輸入值 ($input.<input-name>.<path-to-datum>)。

Amazon CloudWatch 日誌**action.setTimer**錯誤

您可以設定 Amazon CloudWatch 日誌來監控AWS IoT Events偵測器模型執行個體。以下是使用時產生的AWS IoT Events常見錯誤action.setTimer。

- 錯誤：您名為之計時器的持續時間運算式*<timer-name>*無法評估為數字。

解決方案

請確定您的字串運算式durationExpression可以轉換為數字。不允許使用其他資料類型，例如布林值。

- 錯誤：名為之計時器的持續時間運算式的評估結果大*<timer-name>*於 31622440。為了確保準確性，請確保您的持續時間表達式參考的值介於 60—31622400 之間。

解決方案

確保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運算式的評估結果小*<timer-name>*於 60。為了確保準確性，請確保您的持續時間表達式參考的值介於 60—31622400 之間。

解決方案

確保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：無*<timer-name>*法評估名為之計時器的持續時間運算式。檢查變量名稱，輸入名稱和數據的路徑，以確保您參考現有的變量和輸入。

解決方案

確保您的字符串表達式引用了現有的變量和輸入。字串運算式可以包含數字、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>。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 日誌來監控AWS IoT Events偵測器模型執行個體。以下是設定動作承載時所AWS IoT Events產生的常見錯誤和警告。

- 錯誤：我們無法評估您的動作表達方式。請確定變數名稱、輸入名稱和資料路徑參照現有的變數和輸入值。此外，請確認承載的大小小於 1 KB，即承載的允許大小上限。

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，您可能也會收到此錯誤訊息。

- 錯誤：我們無法剖析您的內容運算式以取得的裝載*<action-type>*。使用正確的語法輸入內容表示式。

解決方案

內容運算式可以包含字串 ('*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：您的有效負載*### {expression}* 無效。定義的裝載類型為 JSON，因此您必須指定AWS IoT Events將評估為字串的運算式。

解決方案

如果指定的裝載類型為 JSON，請AWS IoT Events先檢查服務是否可以將運算式評估為字串。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，但我們無法將動作裝載的內容運算式評估為有效的 JSON。定義的裝載類型為 JSON。

解決方案

如果您將裝載類型定義為，請確保AWS IoT Events可以將動作有效負載評估為有效 JSON 的內容表達式JSON。 AWS IoT Events即使無法將內容運算式評估為有效的 JSON，也會執行動作。AWS IoT Events

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。

資料類型不相容

訊息：在下列運算式中找到不相容<reference>的資料類型 [<inferred-types>]：<expression>

解決方案

您可能會因為下列其中一個原因而收到此錯誤：

- 參考的評估結果與運算式中的其他運算元不相容。

- 不支援傳遞給函數的引數類型。

當您在運算式中使用參照時，請檢查下列項目：

- 當您使用具有一或多個運算子的參考做為運算元時，請確定您參考的所有資料型別都相容。

  例如，在下列運算式中，integer 2 是==和&&運算子的運算元。為了確保操作數是兼容的，`$variable.testVariable + 1`並且`$variable.testVariable`必須引用整數或小數。

  此外，整數1是運算子的+運算元。因此，`$variable.testVariable`必須引用整數或小數。

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

- 當您使用參考作為傳遞給函數的引數時，請確保該函數支持您引用的數據類型。

  例如，下列`timeout("time-name")`函數需要以雙引號做為引數的字串。如果您使用`#####`值的參考，則必須使用雙引號引用字串。

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

  > **Note**
  >
  > 對於`convert(type, expression)`函數，如果您使用`##`值的引用，則引用的評估結果必須是`String`、`Decimal`、或`Boolean`。

如需詳細資訊，請參閱 參考。

無法將訊息傳送至 AWS IoT Events

訊息：無法將訊息傳送至 It 事件

解決方案

您可能會因為下列原因而遇到此錯誤：

- 輸入訊息承載不包含`Input attribute Key`.
- 與輸入定義中指定的 JSON 路徑不相同。`Input attribute Key`
- 輸入訊息與結構描述不相符，如AWS IoT Events輸入中所定義。

> **Note**
>
> 從其他服務擷取的資料也會發生失敗。

Example

例如AWS IoT Core，AWS IoT規則將失敗，並顯示下列訊息 `Verify the Input Attribute key.`

若要解決此問題，請確定輸入裝載訊息結構描述符合 In AWS IoT Events put 定義，且`Input attribute Key`位置相符。如需詳細資訊，請參the section called “在導航窗格中創建輸入”閱以瞭解如何定義AWS IoT Events輸入。

透過執行分析來疑難排解偵測器模

AWS IoT Events可以分析您的檢測器模型並生成分析結果，而無需將輸入數據發送到檢測器模型。AWS IoT Events執行本節所述的一系列分析，以檢查您的偵測器型號。此進階疑難排解解決方案也會摘要診斷資訊，包括嚴重性等級和位置，以便您快速找到並修正偵測器模型中的潛在問題。如需有關偵測器模型的診斷錯誤類型和訊息的詳細資訊，請參閱偵測器模型分析與診斷資訊。

您可以使用AWS IoT Events主控台、API、AWS Command Line Interface(AWS CLI) 或 AWSSDK 來檢視偵測器模型分析中的診斷錯誤訊息。

> **Note**
>
> - 您必須先修正所有錯誤，才能發佈偵測器模型。
> - 我們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，檢測器模型可能無法按預期工作。
> - 您可以同時對`RUNNING`狀態進行多達 10 個分析。

若要瞭解如何分析偵測器模型，請參閱分析檢測器模型（控制台）或分析檢測器模型（AWS CLI）。

主題

- 偵測器模型分析與診斷資訊
- 分析檢測器模型（控制台）
- 分析檢測器模型（AWS CLI）

偵測器模型分析與診斷資訊

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，分析結果分為三大類：
  - 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出現問題的欄位。我們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，檢測器模型可能無法按預期工作。
  - Error (ERROR) — 錯誤結果會通知您偵測器模型中發現的問題。 AWS IoT Events當您嘗試發佈偵測器模型時，會自動執行這組分析。您必須先修正所有錯誤，才能發佈偵測器模型。
- 位置 — 包含可用於在分析結果參照的偵測器模型中尋找欄位的資訊。位置通常包括狀態名稱、轉移事件名稱、事件名稱和運算式 (例如`in state TemperatureCheck in onEnter in event Init in action setVariable`)。
- 類型-分析結果的類型。分析類型分為以下幾類：
  - `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。您必須在提供服務的AWS區域中指定與其他AWS服AWS務搭配使用的動作。
  - `service-limits`— 服務配額 (也稱為限制) 是您AWS帳戶的服務資源或作業數目上限或下限。除非另有說明，否則每個配額都是區域特定規定。根據您的業務需求，您可以更新檢測器模型以避免遇到限制或請求增加配額。您可以要求增加某些配額，而其他配額則無法增加。如需詳細資訊，請參閱 配額。
- **`structure`**— 檢測器模型必須具有所有必需的組件，例如狀態，並遵循AWS IoT Events支持的結構。檢測器模型必須至少具有一個狀態和一個條件，用於評估傳入的輸入數據以檢測重要事件。偵測到事件時，偵測器模型會轉換至下一個狀態，並可叫用動作。這些事件稱為轉換事件。轉移事件必須引導下一個狀態才能進入。
- **`expression-syntax`**— AWS IoT Events 提供數種在建立和更新偵測器模型時指定值的方法。您可以在運算式中使用常值、運算子、函數、參照和替代範本。您可以使用運算式來指定常值，或AWS IoT Events在指定特定值之前評估運算式。您的運算式必須遵循所需的語法。如需詳細資訊，請參閱 表達式。

  中的偵測器模型運算式AWS IoT Events可以參考特定資料或資源。
  - **`data-type`**— AWS IoT Events 支援整數、十進位、字串和布林資料類型。如果AWS IoT Events可以在運算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

> **Note**
>
> - 整數和小數是唯一支持的兼容數據類型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符串。

- **`referenced-data`**— 您必須先定義偵測器模型中參照的資料，然後才能使用資料。例如，如果您想要將資料傳送至 DynamoDB 表，您必須先定義參考資料表名稱的變數，才能在運算式 () `$variable.TableName` 中使用變數。
- **`referenced-resource`**— 檢測器模型使用的資源必須可用。您必須先定義資源，才能使用它們。例如，您想要建立偵測器模型來監視溫室的溫度。您必須先定義 input (`$input.TemperatureInput`)，將傳入的溫度資料路由到偵測器模型，然後才能使用`$input.TemperatureInput.sensorData.temperature`來參考溫度。

請參閱以下章節以疑難排解錯誤，並從檢測器模型的分析中找到可能的解決方案。

排除偵測器模型錯誤

上述錯誤類型提供有關偵測器模型的診斷資訊，並與您可能擷取的訊息相對應。使用這些訊息和建議的解決方案，以疑難排解偵測器模型的錯誤。

訊息與解決方案

- Location
- supported-actions
- service-limits
- structure
- expression-syntax
- data-type
- referenced-data
- referenced-resource

`Location`

包含相關`Location`資訊的分析結果會對應於下列錯誤訊息：

- 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：如果您指定的動作AWS IoT Events目前不支援，您可能會收到此錯誤訊息。如需支援動作的清單，請參閱支援的動作。

supported-actions

含有相關supported-actions資訊的分析結果會對應下列錯誤訊息：

- 訊息：動作定義中存在無效的動作類型：*#*作定義。

  解決方案：如果您指定的動作AWS IoT Events目前不支援，您可能會收到此錯誤訊息。如需支援動作的清單，請參閱支援的動作。

- 訊息： DetectorModel 定義具有*aws-service*動作，但區*#*名稱不支援*aws-service*服務。

  解決方案：如果您指定的動作受支援AWS IoT Events，但您目前的區域無法使用該動作，您可能會收到此錯誤訊息。當您嘗試將資料傳送至該地區無法使用的AWS服務時，可能會發生這種情況。您還必須為所使用的AWS服務AWS IoT Events和選擇相同的區域。

service-limits

含有相關service-limits資訊的分析結果會對應下列錯誤訊息：

- 訊息:*##############################content-expression-size####*

  解決方案：如果動作承載的內容運算式大於 1024 位元組，您可能會收到此錯誤訊息。有效負載的內容運算式大小最多可以是 1024 個位元組。

- 訊息：偵測器模型定義中允許的狀態數目超過限制*states-per-detector-model*。

  解決方案：如果您的偵測器型號有超過 20 個狀態，您可能會收到此錯誤訊息。檢測器模型最多可以有 20 個狀態。

- 消息：計時器計時器*###*持續時間至少*minimum-timer-duration*應為秒長。

  解決方案：如果計時器的持續時間少於 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。如果您在計時器的持續時間內指定運算式，則持續時間運算式的評估結果會四捨五入為最接近的整數。

- 訊息：每個事件允許的動作數量超過偵測器模型定義*actions-per-event*中的限制

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。檢測器模型中的每個事件最多可以有 10 個動作。

- 訊息：每個狀態允許的轉換事件數目超過偵測器模型定義*transition-events-per-state*中的限制。

  解決方案：如果狀態有超過 20 個轉換事件，您可能會收到這個錯誤訊息。檢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目超過偵測器模型定義*events-per-state*中的限制

  解決方案：如果狀態有超過 20 個事件，您可能會收到此錯誤訊息。檢測器模型中的每個狀態最多可以有 20 個事件。

- 消息：可以與單個輸入相關聯的檢測器模型的最大數量可能已達到限制。輸入*####*在*detector-models-per-input*檢測器模型路由中使用。

  解決方案：如果您嘗試將輸入路由傳送至 10 個以上的偵測器型號，您可能會收到此警告訊息。您可以有多達 10 個不同的檢測器模型與單個檢測器模型相關聯。

`structure`

含有相關`structure`資訊的分析結果會對應下列錯誤訊息：

- 訊息：動作可能只定義了一個類型，但找到具有*number-of-types*類型的動作。請分成單獨的操作。

  解決方案：如果您使用 API 作業建立或更新偵測器模型，在單一欄位中指定了兩個或多個動作，則可能會收到此錯誤訊息。您可以定義`Action`物件的陣列。請務必將每個動作定義為單獨的物件。

- 訊息： TransitionEventtransition-event-name*###########*

  解決方案：如果找AWS IoT Events不到轉換事件所參考的下一個狀態，您可能會收到此錯誤訊息。請確定已定義下一個狀態，並輸入正確的狀態名稱。

- 消息： DetectorModelDefinition 有一個共享狀態名稱：找到重複的*####*。*number-of-states*

  解決方案：如果您對一或多個狀態使用相同的名稱，您可能會收到此錯誤訊息。確保您為檢測器模型中的每個狀態提供唯一的名稱。狀態名稱必須包含 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*不對應於已定義的狀態。

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。檢測器模型保持在初始（啟動）狀態，直到輸入到達。一旦輸入到達，檢測器模型立即轉換到下一個狀態。請確定初始狀態名稱是已定義狀態的名稱，且您輸入正確的名稱。

- 訊息：偵測器模型定義在條件下必須至少使用一個輸入。

  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下使用一個輸入。否則，AWS IoT Events不會評估傳入的資料。

- 訊息：只能在中設定秒數和持續時間中的一個表示式。 SetTimer

  解決方案：如果您同時使用`seconds`和`durationExpression`計時器，則可能會收到此錯誤訊息。請確定您使用`seconds`或`durationExpression`作為的參數`SetTimerAction`。如需詳細資訊，請參閱 AWS IoT Events API 參考中的 SetTimerAction。

- 訊息：偵測器模型中的動作無法存取。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存取，則事件的條件評估為 false。檢查包含動作的事件的條件，以確保其評估為 true。當事件的條件評估為 true 時，該動作應該變為可訪問。

- 消息：正在讀取輸入屬性，但這可能是由計時器到期引起的。

  解決方案：當發生以下任一情況時，可以讀取輸入屬性的值：

  - 已收到新的輸入值。
  - 當檢測器中的計時器已過期。

  若要確保只有在接收到輸入的新值時才會評估 input 屬性，請在條件中包含對`triggerType(“Message”)`函數的呼叫，如下所示：

  檢測器模型中評估的原始條件：

  ```
  if ($input.HeartBeat.status == “OFFLINE”)
  ```

  將變得類似於以下內容：

  ```
  if ( triggerType("MESSAGE") &&  $input.HeartBeat.status == “OFFLINE”)
  ```

  其中，對`triggerType(“Message”)`函數的調用來自條件中提供的初始輸入之前。通過使用這種技術，該`triggerType("Message")`函數將評估為 true 並滿足接收新輸入值的條件。如需有

關triggerType函數使用方式的詳細資訊，請triggerType在AWS IoT Events開發人員指南的「運算式」區段中搜尋

- 消息：無法訪問檢測器模型中的狀態。檢查將導致過渡到所需狀態的條件。

  解決方案：如果偵測器模型中的某個狀態無法存取，則導致該狀態的傳入轉換的條件會評估為 false。檢查檢測器模型中到該無法訪問狀態的傳入轉換的條件評估為 true，以便所需的狀態可以達到。

- 訊息：即將到期的計時器可能會導致傳送意外數量的訊息。

  解決方案：為了防止您的檢測器模型因計時器已過期而進入無限發送意外數量消息的狀態，請考慮在檢測器模型的條件下使用對triggerType("Message")函數的調用，如下所示：

  檢測器模型中評估的原始條件：

  ```
  if (timeout("awake"))
  ```

  會轉換成類似下列的條件：

  ```
  if (triggerType("MESSAGE") && timeout("awake"))
  ```

  其中，對triggerType(“Message”)函數的調用來自條件中提供的初始輸入之前。

  此更改可防止在檢測器中啟動計時器操作，從而防止發送消息的無限循環。如需有關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開發人員指南的使用內建動作頁面

expression-syntax

含有相關expression-syntax資訊的分析結果會對應下列錯誤訊息：

- 消息：您的有效負載表## {###} 無效。定義的裝載類型為 JSON，因此您必須指定AWS IoT Events將評估為字串的運算式。

  解決方案：如果指定的裝載類型為 JSON，請AWS IoT Events先檢查服務是否可以將運算式評估為字串。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必須是表示式。無法剖析值 '###'

解決方案：您可SetVariableAction以使用name和定義變數value。value可以是字串、數字或布林值。您也可以指定的表示式value。如需詳細資訊 SetVariableAction，請參閱 AWS IoT EventsAPI 參考中的。

- 訊息：我們無法剖析 DynamoDB 動作的屬性 (*####*) 運算式。使用正確的語法輸入表示式。

  解決方案：您必須對中的所有參數使用表示式DynamoDBAction。替代範本。如需詳細資訊，請參閱 API 參考資料中的 AWS IoT EventsDynamoDBAction。

- 訊息：我們無法針對 DynamoDBv2 動作剖析您的 tableName 運算式。使用正確的語法輸入表示式。

  解決方案：tableNamein DynamoDBv2Action 必須是字串。您必須使用的運算式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 API 參考資料中的動作。AWS IoT Events

- 訊息：我們無法將您的運算式評估為有效的 JSON。動作僅支援 JSON 承載類型。

  解決方案：的承載資料類型DynamoDBv2必須是 JSON。確保AWS IoT Events可以評估您的內容表達式的有效載荷為有效的 JSON。如需詳細資訊，請參閱 API 參考資料中的動作。AWS IoT Events

- 消息：我們無法解析您的內容表達式以獲取*####*的有效負載。使用正確的語法輸入內容表示式。

  解決方案：內容表達式可以包含字符串（'*###*'），變量（$ 變量。 *###* ），輸入值（$ 輸入。 *### #。 path-to-datum*）、字串連，以及包含. ${}

- 訊息：自訂承載必須非空白。

  解決方案：如果您為動作選擇了「自訂承載」，但未在AWS IoT Events主控台中輸入內容運算式，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」下輸入內容運算式。如需詳細資訊，請參閱 AWS IoT EventsAPI 參考中的承載。

- 訊息:無法剖析計時器「計時器名稱」的持續時*####*「持續時間表示*#*」。

  解決方案：計時器持續時間運算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:無法剖析動作名稱的*### '###*'

  解決方案：如果指定動作的運算式語法不正確，您可能會收到此訊息。請確定您輸入的表示式的語法正確。如需詳細資訊，請參閱 語法。

- 訊息：IotSitewiseAction無法剖析您的 *fieldName*。您必須在運算式中使用正確的語法。

解決方案：如果AWS IoT Events無法剖析您的 *fieldName*，您可能會收到此錯誤。IotSitewiseAction確保字 *fieldName* 使用可以解析的表達AWS IoT Events式。如需詳細資訊，請參閱 AWS IoT Events API 參考中的 IotSiteWiseAction。

data-type

含有相關data-type資訊的分析結果會對應下列錯誤訊息：

- 消息：計時器計時器定時器*#################*無效，它必須返回一個數字。

  解決方案：如果AWS IoT Events無法將計時器的持續時間運算式評估為數字，您可能會收到此錯誤訊息。確保您durationExpression可以轉換為數字。不支援其他資料類型，例如布林值。

- 訊息：表示式*#####*不是有效的條件運算式。

  解決方案：如果AWS IoT Events無法評估布林值，您condition-expression可能會收到此錯誤訊息。布林值必須是TRUE或FALSE。請確定您的條件運算式可以轉換為布林值。如果結果不是 Boolean 值，則相當於FALSE且不會叫用事件中nextState指定的動作或轉換。

- 訊息：*################ [####] ########*

  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運算式都必須參考相同的資料類型。

  請使用下列資訊來解決問題：

  - 當您使用具有一或多個運算子的參考做為運算元時，請確定您參考的所有資料型別都相容。

    例如，在下列運算式中，integer 2 是==和&&運算子的運算元。為了確保操作數是兼容的，$variable.testVariable + 1並且$variable.testVariable必須引用整數或小數。

    此外，整數1是運算子的+運算元。因此，$variable.testVariable必須引用整數或小數。

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

  - 當您使用參考作為傳遞給函數的引數時，請確保該函數支持您引用的數據類型。

    例如，下列timeout("*time-name*")函數需要以雙引號做為引數的字串。如果您使用*#####*值的參考，則必須使用雙引號引用字串。

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

> **Note**
>
> 對於convert(*type*, *expression*)函數，如果您使用##值的引用，則引用的評估結果必須是StringDecimal、或Boolean。

如需詳細資訊，請參閱 參考。

- 消息：*######## [####] ########*這可能會導致執行階段錯誤。

  解決方案：如果相同輸入屬性或變數的兩個運算式參照兩種資料類型，您可能會收到此警告訊息。確保相同輸入屬性或變量的表達式引用了檢測器模型中的相同數據類型。

- 訊息：*##### [###] ####### [####] ##########'expression'*

  解決方案：如果運算式結合了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，在下列運算式中，運算+子與「整數」、「十進位」和「字串」資料類型相容，但不相容於布林資料類型的運算元。

  ```
  true + false
  ```

  您必須確定搭配運算子使用的資料類型是相容的。

- 訊息：針對*########## [####] ##########*行階段錯誤。

  解決方案：如果相同輸入屬性的兩個運算式針對某個狀態或OnInputLifecycle與OnExitLifecycle狀態的兩個資料類型參考兩OnEnterLifecycle種資料類型，您可能會收到此錯誤訊息。確保OnEnterLifecycle（或，OnInputLifecycle和OnExitLifecycle）中的表達式為檢測器模型的每個狀態引用相同的數據類型。

- 訊息：裝載運算*# [###]* 無效。指定在執行階段評估為字串的運算式，因為承載類型為 JSON 格式。

  解決方案：如果您指定的裝載類型是 JSON，但無法將其運算式評估為 String，則AWS IoT Events可能會收到此錯誤。確保評估的結果是一個字符串，而不是布爾或數字。

- 消息：您的插值表達式 {*#####*} 必須在運行時評估為整數或布爾值。否則，您的有效負載表達式 {有效*#####*} 將無法在運行時解析為有效的 JSON。

  解決方案：如果AWS IoT Events無法將內插運算式評估為整數或布林值，您可能會收到此錯誤訊息。請確定您的內插運算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。

- 消息：**IotSitewiseAction**##########定義的類型和推斷的類型必須相同。

  解決方案：如果 for 中的運算式定義的資料類型與所AWS IoT Events推斷propertyValueIotSitewiseAction的資料類型不同，您可能會收到此錯誤訊息。確保在檢測器模型中對此表達式的所有實例使用相同的數據類型。

- 消息：*##setTimer####### [####] ####Integer##########*

  解決方案：如果持續時間運算式的推斷資料類型不是「整數」或「Decimal」，您可能會收到此錯誤訊息。確保您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：*###### [###] ############ [####] ##############*

  解決方案：*##################################*在檢測器模型的所有其他部分中，操作數必須與匹配的數據類型一起使用。

> **Tip**
>
> 您可以使用convert來變更偵測器模型中運算式的資料類型。如需詳細資訊，請參閱 函數。

referenced-data

含有相關referenced-data資訊的分析結果會對應下列錯誤訊息：

- 訊息：偵測到損壞的計時器：計時器*#####*用於運算式中，但永遠不會設定。

  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。在運算式中使用計時器之前，您必須先設定計時器。另外，請確保輸入正確的計時器名稱。

- 訊息:偵測到損毀的*##############*，但永遠不會設定。

  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。在運算式中使用變數之前，您必須先設定變數。此外，請確定您輸入正確的變數名稱。

- 訊息：偵測到中斷的變數：在設定為值之前，會在運算式中使用變數。

  解決方案：必須將每個變數指派給一個值，才能在運算式中對其進行評估。在每次使用之前設置變量的值，以便可以檢索其值。此外，請確定您輸入正確的變數名稱。

`referenced-resource`

含有相關`referenced-resource`資訊的分析結果會對應下列錯誤訊息：

- 訊息：偵測器模型定義包含不存在之輸入的參照。

  解決方案：如果您使用運算式來參考不存在的輸入，可能會收到此錯誤訊息。請確定您的運算式參考現有的輸入，並輸入正確的輸入名稱。如果您沒有輸入，請先創建一個輸入。

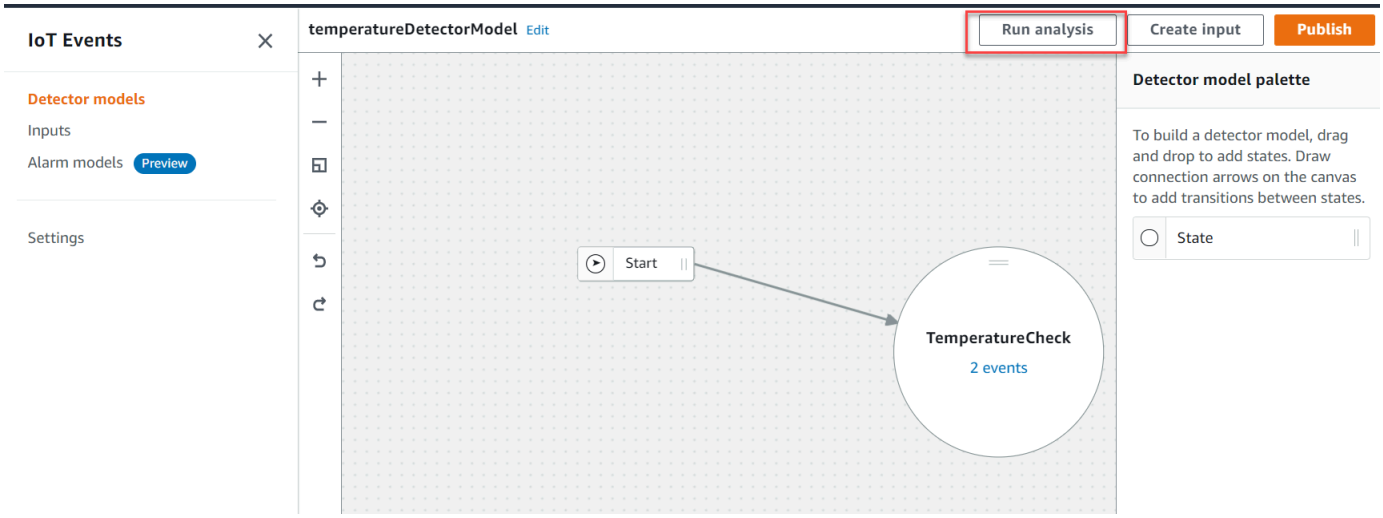
- 訊息：偵測器模型定義包含無效 InputName：##名稱

  解決方案：如果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。確保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)。

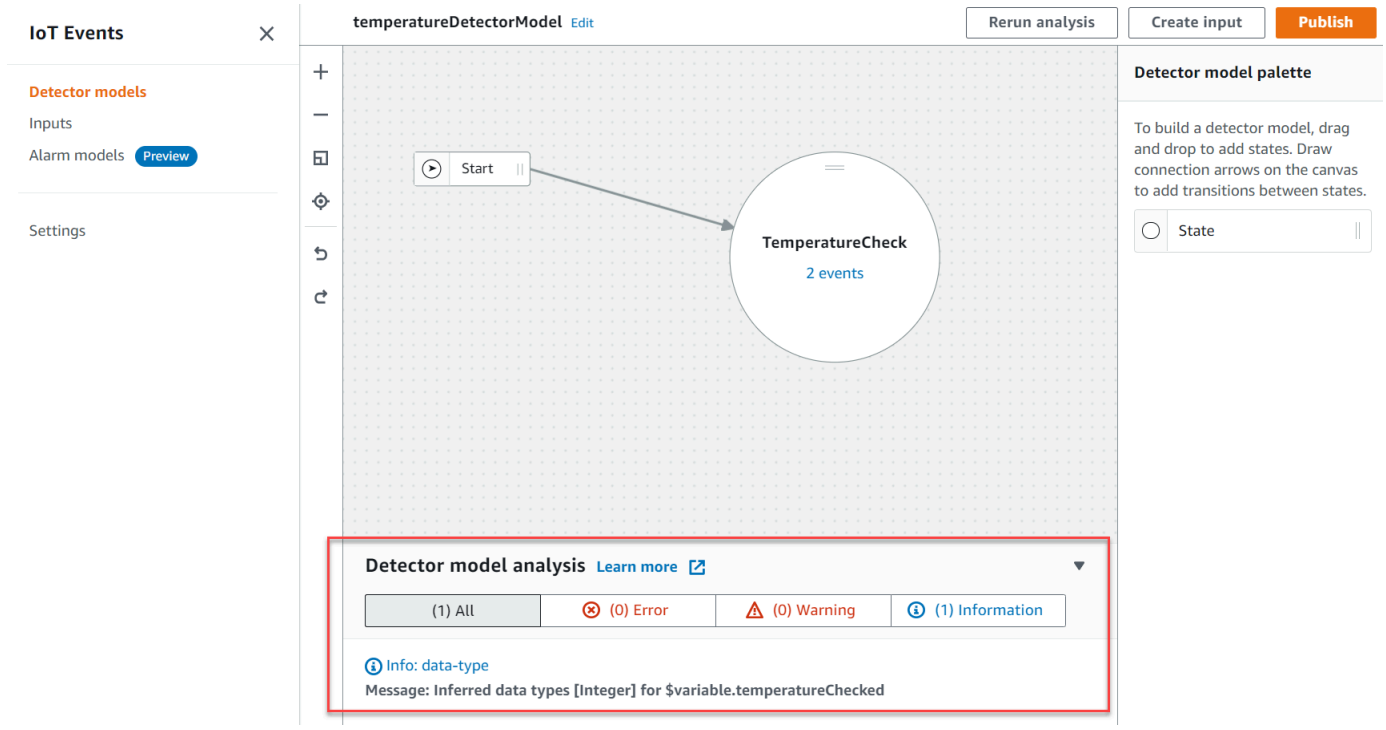
分析檢測器模型（控制台）

以下步驟使用AWS IoT Events控制台分析檢測器模型。

1. 登入 AWS IoT Events 主控台。
2. 在導覽窗格中，選擇 [偵測器型號]。
3. 在偵測器型號下，選擇目標偵測器型號。
4. 在偵測器型號頁面上，選擇 [編輯]。
5. 在右上角，選擇 [執行分析]。



以下是AWS IoT Events控制台中的分析結果示例。



> **Note**
>
> AWS IoT Events開始分析您的偵測器模型後，您最多有 24 小時的時間來擷取分析結果。

分析檢測器模型（AWS CLI）

下列步驟使用AWS CLI來分析偵測器模型。

1. 執行下列指令以開始分析。

```
aws iotevents start-detector-model-analysis --cli-input-json file://file-name.json
```

> **Note**
>
> 將檔######包含偵測器模型定義的檔案名稱。

Example 檢測器型號定義

```
{
    "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：

- 如果要使用AWS IoT Events控制台，請執行以下操作：
  1. 在導覽窗格中，選擇 [偵測器型號]。
  2. 在偵測器型號下，選擇目標偵測器型號。
  3. 從中選擇導出檢測器模型行動下載檢測器模型。檢測器模型保存在 JSON 中。
  4. 開啟偵測器模型 JSON 檔案。
  5. 你只需要對`detectorModelDefinition`象。移除下列項目：
     - 頁面頂端的第一個大括號 ({)
     - 該`detectorModel`行
     - 該`detectorModelConfiguration`對象
     - 頁面底部的最後一個大括號 (})
  6. 儲存檔案。
- 如果您要使用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. 替換*detector-model-name*為檢測器型號的名稱。
  3. 將`detectorModelDefinition`物件複製到文字編輯器。
  4. 在之外加入大括號 ({}) `detectorModelDefinition`。
  5. 將檔案儲存為 JSON 格式。

Example 回應範例

```
{
    "analysisId": "c1133390-14e3-4204-9a66-31efd92a4fed"
}
```

2. 從輸出複製分析 ID。
3. 執行下列命令以擷取分析狀態。

```
aws iotevents describe-detector-model-analysis --analysis-id "analysis-id"
```

> **Note**
>
> 將分####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 正在分析您的檢測器模型。此程序最多可能需要一分鐘才能完成。
- COMPLETE— AWS IoT Events 完成分析您的探測器模型。
- FAILED— AWS IoT Events 無法分析您的檢測器模型。請稍後再試。

4. 執行下列指令以擷取偵測器模型的一或多個分析結果。

> **Note**
>
> 將分####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開始分析您的偵測器模型後，您最多有 24 小時的時間來擷取分析結果。

AWS IoT Events 命令

本章將引導您詳細介紹所有 API 操作，包括支持的 Web 服務協議的示例請求，響應和錯誤。AWS IoT Events

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 EventsAPI 參考AWS IoT Events中支援的動作和資料類型。

《AWS CLI指AWS CLI令參考》中的AWS IoT Events各節包括可用於管理和操控的指令AWS IoT Events。

AWS IoT Events 資料

您可以使用 AWS IoT Events Data API 命令將輸入傳送至偵測器、清單偵測器，以及檢視或更新偵測器的狀態。如需詳細資訊，請參閱 AWS IoT EventsAPI 參考中的資AWS IoT Events料支援的動作和資料類型。

《指AWS CLI令參考》中的AWS IoT Events資料區段包括可用於處理AWS IoT Events資料的AWS CLI指令。

文件歷史紀錄

下表說明 2020 年 9 月 17 日之後對開AWS IoT Events發人員指南所做的重要變更。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
區域啟動	AWS IoT Events 現已在亞太地區 (孟買) 區域提供。	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
透過執行分析疑難排解偵測器模型	AWS IoT Events現在可以分析您的偵測器模型並產生分析結果，您可以用來排除偵測器模型的疑難排解。	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
運算式用法	已新增範例，向您展示如何撰寫運算式。	2020 年 9 月 22 日
使用警報監控	警示可協助您監控資料是否有變更。您可以建立警示，以便在超出閾值時傳送通知。	2020 年 6 月 1 日

舊版更新

下表說明 2020 年 9 月 18 日之前對AWS IoT Events開發人員指南進行的重要變更。

變更	描述	日期
新增	添加了類型驗證參考。	2020 年 8 月 3 日

變更	描述	日期
新增	已將區域資訊新增至與其他 AWS 服務合作。	2020 年 5 月 7 日
新增、更新	已新增承載自訂功能和新的事件動作：Amazon DynamoDB 和. AWS IoT SiteWise	2020 年 4 月 27 日
編輯	已新增狀態機概念的新描述。內容的一般編輯。	2019 年 10 月 31 日
新增	為檢測器模型條件表達式添加了新的內置函數。	2019 年 9 月 10 日
新增	新增偵測器模型範例。	2019 年 8 月 5 日
新增	新增了新的事件動作：Lambda、Amazon SQS、Kinesis Data Firehose 和輸入。AWS IoT Events	2019 年 7 月 19 日
補充，更正	更正`timeout()` 函數的描述。已新增關於帳戶閒置的最佳做法。	2019 年 6 月 11 日
更正	更新：控制台調試選項頁面圖像; 控制台權限策略。	2019 年 6 月 5 日
更新	AWS IoT Events服務對正式可用性開放。	2019 年 5 月 30 日
新增、更新	已更新安全性資訊；新增附註的偵測器模型範例。	2019 年 5 月 22 日
更正	已更新：限制預覽下載的連結；Amazon SNS 承載資料範例；新增的必要許可 CreateDetectorModel。	2019 年 5 月 17 日

變更	描述	日期
新增	已新增安全性相關資訊。	2019 年 5 月 9 日
更正	鏈接到有限的預覽下載更正。	2019 年 4 月 19 日
編輯	編輯精選的改進功能。	2019 年 4 月 16 日
有限的預覽版	文件的有限預覽版本。	2019 年 3 月 28 日
編輯	編輯精選的改進功能。	2018年5月18日

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。