



Guía para desarrolladores

AWS IoT Events



AWS IoT Events: Guía para desarrolladores

Copyright © 2025 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Las marcas registradas y la imagen comercial de Amazon no se pueden utilizar en relación con ningún producto o servicio que no sea de Amazon de ninguna forma que pueda causar confusión entre los clientes y que menosprecie o desacredite a Amazon. Todas las demás marcas registradas que no son propiedad de Amazon son propiedad de sus respectivos propietarios, que pueden o no estar afiliados, conectados o patrocinados por Amazon.

Table of Contents

.....	viii
¿Qué es AWS IoT Events?	1
Beneficios y características	1
Casos de uso	3
Supervise y mantenga los dispositivos remotos	3
Administre robots industriales	3
Rastrea los sistemas de automatización de edificios	3
AWS IoT Events fin del soporte	4
Consideraciones a la hora de migrar desde AWS IoT Events	4
Modelos de detector	5
Comparación de arquitecturas	5
Paso 1: (opcional) exportar AWS IoT Events las configuraciones del modelo de detector	7
Paso 2: creación de un rol de IAM	8
Paso 3: Crear Amazon Kinesis Data Streams	10
Paso 4: Cree o actualice la regla de enrutamiento de mensajes de MQTT	11
Paso 5: Obtenga el punto final del tema MQTT de destino	12
Paso 6: Crear una tabla de Amazon DynamoDB	13
Paso 7: Crear una AWS Lambda función (consola)	14
Paso 8: Añadir un activador de Amazon Kinesis Data Streams	22
Paso 9: Pruebe la funcionalidad de entrada y salida de datos (AWS CLI)	23
Alarmas	23
Comparación de arquitecturas	24
Paso 1: Habilite las notificaciones MQTT en la propiedad del activo	24
Paso 2: Crear una función de AWS Lambda	25
Paso 3: Crea AWS IoT Core una regla de enrutamiento de mensajes	27
Paso 4: Ver CloudWatch las métricas	28
Paso 5: Crear CloudWatch alarmas	29
Paso 6: (opcional) importe la CloudWatch alarma a AWS IoT SiteWise	29
Configuración	30
Configurar un Cuenta de AWS	30
Inscríbase en un Cuenta de AWS	30
Creación de un usuario con acceso administrativo	31
Configuración de permisos para AWS IoT Events	32
Permisos de acción	33

Protección de los datos de entrada	35
Política de funciones de CloudWatch registro de Amazon	35
Política de roles de mensajería de Amazon SNS	37
Introducción	39
Requisitos previos	41
Crear una entrada	42
Cree un archivo de entrada JSON	42
Cree y configure una entrada	42
Cree una entrada dentro del modelo de detector	43
Crear un modelo de detector	44
Pruebe el modelo de detector	51
Prácticas recomendadas	55
Habilite el CloudWatch registro de Amazon al desarrollar modelos AWS IoT Events de detectores	55
Publique periódicamente para guardar su modelo de detector cuando trabaje en la AWS IoT Events consola	56
Tutoriales	57
Uso AWS IoT Events para monitorear sus dispositivos de IoT	57
¿Cómo puede saber qué estados necesita en un modelo de detector?	59
¿Cómo puede saber si necesita una instancia, o varias, de un detector?	60
step-by-stepEjemplo sencillo	61
Creación de una entrada para capturar datos del dispositivo	63
Creación de un modelo de detector para representar los estados de los dispositivos	64
Envío de mensajes como entradas a un detector	68
Restricciones y limitaciones del modelo de detector	71
Un ejemplo comentado: Control de temperatura de un climatizador	75
Definiciones de entrada para modelos de detectores	76
Cree una definición de modelo de detector	79
Uso de BatchUpdateDetector	99
BatchPutMessage Úselo para entradas	102
Ingiera mensajes MQTT	104
Generar mensajes de Amazon SNS	105
Configure la DescribeDetector API	106
Utilice el motor de AWS IoT Core reglas	109
Acciones admitidas	112
Usa las acciones integradas	113

Establecer la acción del temporizador	113
Restablecer la acción de temporizador	114
Eliminar la acción del temporizador	114
Establecer una acción de variable	115
Trabaja con otros AWS servicios	115
AWS IoT Core	116
AWS IoT Events	117
AWS IoT SiteWise	118
Amazon DynamoDB	121
Amazon DynamoDB(v2)	123
Amazon Data Firehose	124
AWS Lambda	126
Amazon Simple Notification Service	127
Amazon Simple Queue Service	128
Expressions	130
Sintaxis para filtrar los datos del dispositivo	130
Literales	130
Operadores	130
Funciones para expresiones	132
Referencia para las entradas y variables en las expresiones	137
Plantillas de sustitución	140
Uso	140
Escribir AWS IoT Events expresiones	141
Ejemplos de modelos de detectores	143
Control de temperatura de climatización	143
Antecedentes	143
Definiciones de entradas	144
Definición del modelo de detector	147
BatchPutMessage ejemplos	164
BatchUpdateDetector ejemplo	170
AWS IoT Core motor de reglas	172
Grúas	175
Enviar comandos	176
Modelos de detector	178
Entradas	184
Mensajes	185

Ejemplo: detección de eventos con sensores	187
Dispositivo HeartBeat	189
Alarmas de ISA	191
Alarma sencilla	201
Monitoreo con alarmas	206
Trabajando con AWS IoT SiteWise	206
Flujo de reconocimiento	206
Creación de un modelo de alarma	207
Requisitos	208
Creación de un modelo de alarma (consola)	208
Respuesta a las alarmas	211
Administración de las notificaciones de alarma	213
Creación de una función de Lambda	213
Uso de la función Lambda	222
Gestione los destinatarios de las alarmas	223
Seguridad	225
Identity and Access Management	225
Público	226
Autenticación con identidades	227
Administración de acceso mediante políticas	230
Más información sobre la gestión de identidades y accesos	232
¿Cómo AWS IoT Events funciona con IAM	232
Ejemplos de políticas basadas en identidades	237
Prevención policial confusa entre servicios para AWS IoT Events	243
Solución de problemas	247
Monitorización	249
Herramientas disponibles para monitorear AWS IoT Events	250
Monitorización AWS IoT Events con Amazon CloudWatch	251
Registrar llamadas a AWS IoT Events la API con AWS CloudTrail	253
Validación de conformidad	273
Resiliencia	274
Seguridad de la infraestructura	274
Cuotas	276
Etiquetado	277
Conceptos básicos de etiquetas	277
Restricciones y limitaciones en las etiquetas	278

Uso de etiquetas con políticas de IAM	279
Solución de problemas	282
AWS IoT Events Problemas y soluciones comunes	282
Errores de creación del modelo de detector	283
Actualizaciones de un modelo de detector eliminado	283
Fallo de activación de la acción (al cumplirse una condición)	283
Fallo de activación de la acción (al superar un umbral)	284
Uso incorrecto de los estados	284
Mensaje de conexión	284
InvalidRequestException mensaje	285
action.setTimerErrores de Amazon CloudWatch Logs	285
Errores de CloudWatch carga útil de Amazon	286
Tipos de datos incompatibles	288
No se pudo enviar el mensaje a AWS IoT Events	289
Solución de problemas de un modelo de detector	290
Información de diagnóstico	291
Analice un modelo de detector (consola)	305
Analice un modelo de detector (AWS CLI)	306
Comandos	312
AWS IoT Events acciones	312
AWS IoT Events datos	312
Historial del documento	313
Actualizaciones anteriores	314

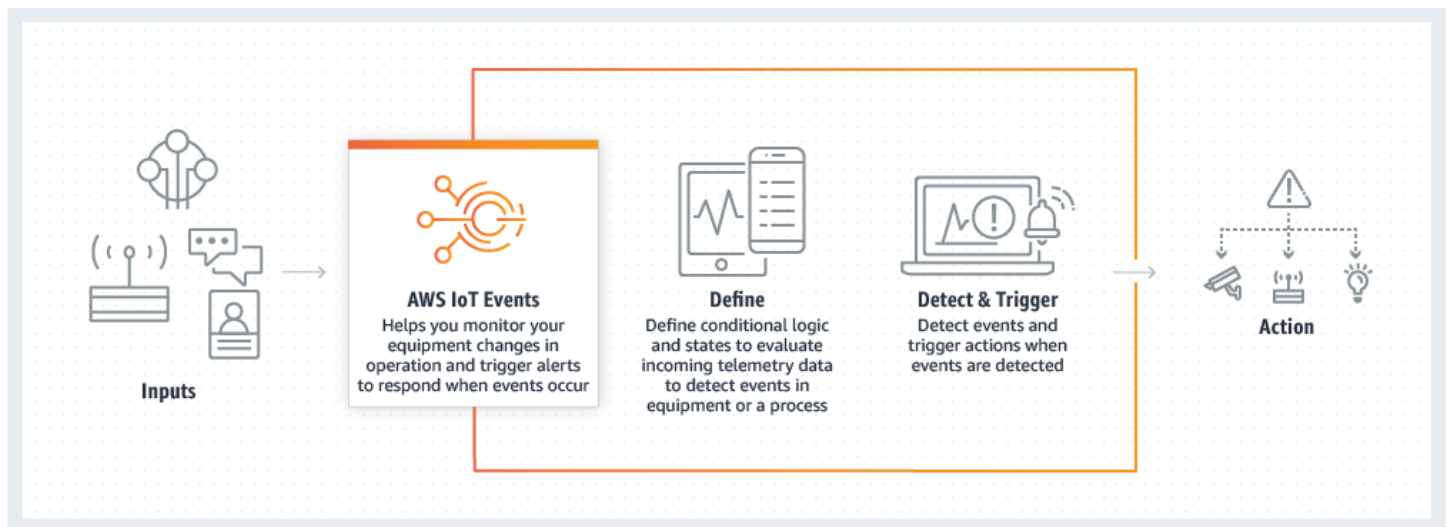
Aviso de fin de soporte: el 20 de mayo de 2026, AWS finalizará el soporte para AWS IoT Events. Después del 20 de mayo de 2026, ya no podrás acceder a la AWS IoT Events consola ni a AWS IoT Events los recursos. Para obtener más información, consulta [AWS IoT Events el fin del soporte](#).

Las traducciones son generadas a través de traducción automática. En caso de conflicto entre la traducción y la version original de inglés, prevalecerá la version en inglés.

¿Qué es AWS IoT Events?

AWS IoT Events le permite monitorizar sus equipos o flotas de dispositivos para detectar fallos o cambios en el funcionamiento y activar acciones cuando se producen dichos eventos. AWS IoT Events monitorea continuamente los datos de los sensores de IoT de los dispositivos, procesos, aplicaciones y otros AWS servicios para identificar eventos importantes y poder tomar medidas.

Úselo AWS IoT Events para crear aplicaciones complejas de monitoreo de eventos en la AWS nube a las que pueda acceder a través de la AWS IoT Events consola o APIs.



Temas

- [Beneficios y características](#)
- [Casos de uso](#)

Beneficios y características

Acepte entradas de múltiples fuentes

AWS IoT Events acepta entradas de muchas fuentes de datos de telemetría de IoT. Estos incluyen dispositivos sensores, aplicaciones de administración y otros AWS IoT servicios, como AWS IoT Core y AWS IoT Analytics. Puede enviar cualquier entrada de datos de telemetría AWS IoT Events mediante una interfaz API (BatchPutMessageAPI) estándar o la AWS IoT Events consola.

Para obtener más información sobre cómo empezar AWS IoT Events, consulte [Primeros pasos con la AWS IoT Events consola](#)

Utilice expresiones lógicas sencillas para reconocer patrones complejos de eventos

AWS IoT Events puede reconocer patrones de eventos que involucran múltiples entradas desde un solo dispositivo o aplicación de IoT, o desde diversos equipos y muchos sensores independientes. Esto es de especial utilidad porque cada sensor y aplicación proporciona información importante. Pero solo combinando diversos datos de sensores y aplicaciones puede obtener una imagen completa del rendimiento y la calidad de las operaciones. Puede configurar AWS IoT Events los detectores para que reconozcan estos eventos mediante expresiones lógicas sencillas en lugar de códigos complejos.

Para obtener más información sobre las expresiones lógicas, consulte [Expresiones para filtrar, transformar y procesar datos de eventos](#).

Activa acciones en función de eventos

AWS IoT Events le permite activar acciones directamente en Amazon Simple Notification Service (Amazon SNS), Lambda AWS IoT Core, Amazon SQS y Amazon Kinesis Firehose. También puede activar una AWS Lambda función mediante el motor de AWS IoT reglas, lo que permite realizar acciones mediante otros servicios, como Amazon Connect, o sus propias aplicaciones de planificación de recursos empresariales (ERP).

AWS IoT Events incluye una biblioteca prediseñada de acciones que puede realizar y también le permite definir las suyas propias.

Para obtener más información sobre cómo activar acciones en función de eventos, consulte [Acciones compatibles para recibir datos y activar acciones en AWS IoT Events](#)

Escale automáticamente para satisfacer las demandas de su flota

AWS IoT Events se escala automáticamente cuando se conectan dispositivos homogéneos. Puede definir un detector una vez para un tipo específico de dispositivo, y el servicio escalará y gestionará automáticamente todas las instancias de ese dispositivo a las que se conecte AWS IoT Events.

Para ver ejemplos de modelos de detectores, consulte [AWS IoT Events ejemplos de modelos de detectores](#).

Casos de uso

AWS IoT Events tiene muchos usos. Estos son algunos ejemplos de casos de uso.

Supervise y mantenga los dispositivos remotos

Supervisar una flota de máquinas desplegadas de forma remota puede resultar difícil, especialmente cuando se produce un mal funcionamiento sin un contexto claro. Si una máquina deja de funcionar, puede ser necesario sustituir toda la unidad de procesamiento o la máquina. Pero esto no es sostenible. Con AWS IoT Events él, puede recibir mensajes de varios sensores en cada máquina para ayudarlo a diagnosticar problemas específicos a lo largo del tiempo. En lugar de sustituir toda la unidad, ahora dispone de la información necesaria para enviar a un técnico la pieza exacta que necesita sustituir. Con millones de máquinas, el ahorro podría ascender a millones de dólares, reduciendo sus costos totales de propiedad o mantenimiento de cada máquina.

Administre robots industriales

La implementación de robots en sus instalaciones para automatizar el movimiento de paquetes puede mejorar considerablemente la eficiencia. Para minimizar los costes, los robots pueden equiparse con sensores sencillos y de bajo coste que envíen los datos a la nube. Sin embargo, con docenas de sensores y cientos de modos de funcionamiento, detectar problemas en tiempo real puede resultar difícil. Con AWS IoT Events él, puede crear un sistema experto que procese los datos de los sensores en la nube y cree alertas para notificar automáticamente al personal técnico en caso de que se produzca un fallo inminente.

Rastrea los sistemas de automatización de edificios

En los centros de datos, el monitoreo de las altas temperaturas y la baja humedad ayuda a prevenir fallas en los equipos. Los sensores suelen adquirirse de muchos fabricantes y cada tipo viene con su propio software de gestión. Sin embargo, el software de administración de diferentes proveedores a veces no es compatible, lo que dificulta la detección de problemas. Con AWS IoT Events él, puede configurar alertas para notificar a sus analistas de operaciones los problemas en sus sistemas de calefacción y refrigeración con bastante antelación a las averías. De este modo, puede evitar un cierre no programado del centro de datos que le costaría miles de dólares en sustitución de equipos y una posible pérdida de ingresos.

AWS IoT Events fin del soporte

Tras considerarlo detenidamente, decidimos poner fin al soporte del AWS IoT Events servicio a partir del 20 de mayo de 2026. AWS IoT Events ya no aceptará nuevos clientes a partir del 20 de mayo de 2025. Como cliente actual con una cuenta registrada en el servicio antes del 20 de mayo de 2025, puedes seguir utilizando AWS IoT Events las funciones. Después del 20 de mayo de 2026, ya no podrás utilizarlas AWS IoT Events.

Esta página proporciona instrucciones y consideraciones para que AWS IoT Events los clientes hagan la transición a una solución alternativa que satisfaga sus necesidades empresariales.

Note

Las soluciones que se presentan en estas guías pretenden servir como ejemplos ilustrativos, no como sustitutos de la funcionalidad listos para la producción. AWS IoT Events Personalice el código, el flujo de trabajo y los AWS recursos relacionados según las necesidades de su empresa.

Temas

- [Consideraciones a la hora de migrar desde AWS IoT Events](#)
- [Procedimiento de migración de modelos de detectores en AWS IoT Events](#)
- [Procedimiento de migración de AWS IoT SiteWise alarmas en AWS IoT Events](#)

Consideraciones a la hora de migrar desde AWS IoT Events

- Implemente las mejores prácticas de seguridad, como el uso de las funciones de IAM con los privilegios mínimos para cada componente y el cifrado de los datos en reposo y en tránsito. Para más información, consulte [Prácticas recomendadas de seguridad en IAM](#) en la Guía del usuario de IAM.
- Tenga en cuenta el número de fragmentos de la transmisión de Kinesis en función de sus requisitos de ingesta de datos. Para obtener más información sobre los fragmentos de Kinesis, consulte la [terminología y los conceptos de Amazon Kinesis Data Streams](#) en la Guía para desarrolladores de Amazon Kinesis Data Streams.

- Configure una supervisión y una depuración integrales mediante CloudWatch métricas y registros. Para obtener más información, consulta [¿Qué es? CloudWatch](#) en la Guía del CloudWatch usuario de Amazon.
- Tenga en cuenta la estructura de su gestión de errores, incluida la forma de gestionar los mensajes que no se procesan repetidamente, implementar políticas de reintentos y configurar un proceso para aislar y analizar los mensajes problemáticos.
- Usa la [calculadora AWS de precios](#) para estimar los costos de tu caso de uso específico.

Procedimiento de migración de modelos de detectores en AWS IoT Events

En esta sección se describen soluciones alternativas que ofrecen una funcionalidad similar a la de un modelo de detector a medida que se migra AWS IoT Events.

Puede migrar la ingesta de datos mediante AWS IoT Core reglas a una combinación de otros AWS servicios. En lugar de la ingesta de datos a través de la [BatchPutMessageAPI](#), los datos se pueden dirigir al tema MQTT. AWS IoT Core

Este enfoque de migración aprovecha los temas de AWS IoT Core MQTT como punto de entrada para sus datos de IoT, sustituyendo la entrada directa a. AWS IoT Events Los temas de MQTT se eligen por varios motivos clave. Ofrecen una amplia compatibilidad con los dispositivos de IoT debido al uso generalizado de MQTT en la industria. Estos temas permiten gestionar grandes volúmenes de mensajes procedentes de numerosos dispositivos, lo que garantiza la escalabilidad. También proporcionan flexibilidad a la hora de enrutar y filtrar los mensajes según el contenido o el tipo de dispositivo. Además, los temas de AWS IoT Core MQTT se integran perfectamente con otros AWS servicios, lo que facilita el proceso de migración.

Los datos fluyen de los temas de MQTT a una arquitectura que combina Amazon Kinesis Data Streams, AWS Lambda una función, una tabla de Amazon DynamoDB y las planificaciones de Amazon. EventBridge Esta combinación de servicios replica y mejora la funcionalidad que anteriormente ofrecía AWS IoT Events, ofreciéndole más flexibilidad y control sobre su proceso de procesamiento de datos de IoT.

Comparación de arquitecturas

La AWS IoT Events arquitectura actual ingiere datos a través de una AWS IoT Core regla y la BatchPutMessage API. Esta arquitectura se utiliza AWS IoT Core para la ingesta de datos y

la publicación de eventos, con mensajes enrutados a través de AWS IoT Events las entradas a modelos de detección que definen la lógica de estado. Un rol de IAM administra los permisos necesarios.

La nueva solución se ocupa de AWS IoT Core la ingesta de datos (ahora con temas específicos de MQTT de entrada y salida). Presenta Kinesis Data Streams para el particionamiento de datos y una función Lambda evaluadora para la lógica de estados. Los estados de los dispositivos ahora se almacenan en una tabla de DynamoDB y una función de IAM mejorada administra los permisos en estos servicios.

Finalidad	Solución	Diferencias
Ingesta de datos: recibe datos de dispositivos de IoT	AWS IoT Core	Ahora requiere dos temas distintos de MQTT: uno para la ingesta de datos del dispositivo y otro para la publicación de los eventos de salida
Dirección de los mensajes: enruta los mensajes entrantes a los servicios correspondientes	AWS IoT Core regla de enrutamiento de mensajes	Mantiene la misma funcionalidad de enrutamiento, pero ahora dirige los mensajes a Kinesis Data Streams en lugar de AWS IoT Events
Procesamiento de datos: gestiona y organiza los flujos de datos entrantes	Kinesis Data Streams	Sustituye la funcionalidad AWS IoT Events de entrada y permite la ingesta de datos mediante la partición de la ID del dispositivo para el procesamiento de los mensajes
Evaluación lógica: procesa los cambios de estado y desencadena acciones	Evaluator Lambda	Sustituye al modelo de AWS IoT Events detector y proporciona una evaluación de la lógica de estado personalizable mediante código en lugar de un flujo de trabajo visual
Administración del estado: mantiene los estados de los dispositivos	Tabla de DynamoDB	Nuevo componente que proporciona un almacenamiento persistente de los estados de los dispositivos, en sustitución de la administración interna AWS IoT Events del estado

Finalidad	Solución	Diferencias
Seguridad: administrar a los permisos de servicio	rol de IAM	Los permisos actualizados ahora incluyen el acceso a Kinesis Data Streams y DynamoDB, EventBridge además de los permisos existentes AWS IoT Core

Paso 1: (opcional) exportar AWS IoT Events las configuraciones del modelo de detector

Antes de crear nuevos recursos, exporte las definiciones AWS IoT Events del modelo de detector. Estos contienen su lógica de procesamiento de eventos y pueden servir como referencia histórica para implementar su nueva solución.

Console

Con ellas AWS IoT Events AWS Management Console, lleve a cabo los siguientes pasos para exportar las configuraciones del modelo de su detector:

Para exportar modelos de detectores mediante el AWS Management Console

1. Inicie sesión en la [consola de AWS IoT Events](#).
2. En el panel de navegación izquierdo, elija Detector models (Modelos de detector).
3. Seleccione el modelo de detector que desee exportar.
4. Seleccione Exportar. Lea el mensaje de información sobre la salida y, a continuación, vuelva a seleccionar Exportar.
5. Repita el proceso para cada modelo de detector que desee exportar.

Se agrega un archivo que contiene una salida JSON de su modelo de detector a la carpeta de descargas de su navegador. Si lo desea, puede guardar la configuración de cada modelo de detector para conservar los datos históricos.

AWS CLI

Con el AWS CLI, ejecute los siguientes comandos para exportar las configuraciones del modelo de detector:

Para exportar modelos de detectores mediante AWS CLI

1. Enumere todos los modelos de detectores de su cuenta:

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

2. Para cada modelo de detector, exporte su configuración ejecutando:

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

3. Guarde la salida de cada modelo de detector.

Paso 2: creación de un rol de IAM

Cree un rol de IAM para proporcionar permisos para replicar la funcionalidad de AWS IoT Events. El rol de este ejemplo concede acceso a DynamoDB para la administración del estado, la programación EventBridge, y a Kinesis Data Streams para la AWS IoT Core ingesta de datos, la publicación de mensajes y el registro. CloudWatch En conjunto, estos servicios funcionarán como sustitutos. AWS IoT Events

1. Cree un rol de IAM con los siguientes permisos. Para obtener instrucciones más detalladas sobre la creación de un rol de IAM, consulte [Crear un rol para delegar permisos a un AWS servicio](#) en la Guía del usuario de IAM.

```
{  
  "Version": "2012-10-17",  
  "Statement": [  
    {  
      "Sid": "DynamoDBAccess",  
      "Effect": "Allow",  
      "Action": [  
        "dynamodb:GetItem",  
        "dynamodb:PutItem",  
        "dynamodb:UpdateItem",  
        "dynamodb>DeleteItem",  
        "dynamodb:Query",  
        "dynamodb:Scan"  
      ],  
      "Resource": "arn:aws:dynamodb:your-region:your-account-id:table/  
EventsStateTable"
```

```
    },
    {
      "Sid": "SchedulerAccess",
      "Effect": "Allow",
      "Action": [
        "scheduler:CreateSchedule",
        "scheduler:DeleteSchedule"
      ],
      "Resource": "arn:aws:scheduler:your-region:your-account-id:schedule/*"
    },
    {
      "Sid": "KinesisAccess",
      "Effect": "Allow",
      "Action": [
        "kinesis:GetRecords",
        "kinesis:GetShardIterator",
        "kinesis:DescribeStream",
        "kinesis:ListStreams"
      ],
      "Resource": "arn:aws:kinesis:your-region:your-account-id:stream/*"
    },
    {
      "Sid": "IoTPublishAccess",
      "Effect": "Allow",
      "Action": "iot:Publish",
      "Resource": "arn:aws:iot:your-region:your-account-id:topic/*"
    },
    {
      "Effect": "Allow",
      "Action": "logs:CreateLogGroup",
      "Resource": "arn:aws:logs:your-region:your-account-id:*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "logs:CreateLogStream",
        "logs:PutLogEvents"
      ],
      "Resource": [
        "arn:aws:logs::your-account-id:log-group:/aws/lambda/your-lambda:"
      ]
    }
  ]
}
```

```
}
```

2. Añada la siguiente política de confianza de roles de IAM. Una política de confianza permite a AWS los servicios especificados asumir la función de IAM para poder realizar las acciones necesarias. Para obtener instrucciones más detalladas sobre cómo crear una política de confianza de IAM, consulte [Crear un rol mediante políticas de confianza personalizadas](#) en la Guía del usuario de IAM.

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

Paso 3: Crear Amazon Kinesis Data Streams

Cree Amazon Kinesis Data Streams con AWS Management Console o. AWS CLI

Console

Para crear una transmisión de datos de Kinesis mediante el AWS Management Console, siga el procedimiento que se encuentra en la página [Crear una transmisión de datos de la Guía para desarrolladores de Amazon Kinesis Data Streams](#).

Ajuste el recuento de fragmentos en función del número de dispositivos y del tamaño de la carga útil de los mensajes.

AWS CLI

Utilice AWS CLI Amazon Kinesis Data Streams para incorporar y particionar los datos de sus dispositivos.

En esta migración se utiliza Kinesis Data Streams para reemplazar la funcionalidad de ingesta de datos de AWS IoT Events. Proporciona una forma escalable y eficiente de recopilar, procesar y analizar datos de streaming en tiempo real desde sus dispositivos de IoT, a la vez que proporciona un manejo flexible de los datos y la integración con otros AWS servicios.

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

Ajusta el número de fragmentos en función del número de dispositivos y del tamaño de la carga útil de los mensajes.

Paso 4: Cree o actualice la regla de enrutamiento de mensajes de MQTT

Puede crear una nueva regla de enrutamiento de mensajes de MQTT o actualizar una regla existente.

Console

1. Determine si necesita una nueva regla de enrutamiento de mensajes de MQTT o si puede actualizar una regla existente.
2. Abra la [consola de AWS IoT Core](#).
3. En el panel de navegación, elija Enrutamiento de mensajes y, a continuación, Reglas.
4. En la sección Administrar, selecciona Enrutamiento de mensajes y, a continuación, Reglas.
5. Seleccione Creación de regla.
6. En la página Especificar las propiedades de la regla, introduzca el nombre de la AWS IoT Core regla para Nombre de la regla. En Descripción de la regla (opcional), introduzca una descripción para identificar que está procesando eventos y reenviándolos a Kinesis Data Streams.
7. En la página Configurar una sentencia SQL, introduzca lo siguiente para la sentencia SQL: y, a continuación **SELECT * FROM 'your-database'**, seleccione Siguiente.
8. En la página Adjuntar reglas y acciones y, en Acciones de regla, elija kinesis.
9. Elija su transmisión de Kinesis para la transmisión. Escriba **your-instance-id** como clave de partición. Seleccione la función adecuada para la función de IAM y, a continuación, elija Añadir acción de regla.

Para obtener más información, consulte [Crear reglas de AWS IoT para enrutar los datos del dispositivo a otros servicios](#).

AWS CLI

1. Cree un archivo JSON denominado con el siguiente contenido. Este archivo de configuración JSON define una AWS IoT Core regla que selecciona todos los mensajes de un tema y los reenvía a la transmisión de Kinesis especificada, utilizando el ID de instancia como clave de partición.

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

2. Cree la regla temática de MQTT con. AWS CLI En este paso se utiliza AWS CLI para crear una regla AWS IoT Core temática con la configuración definida en el `events_rule.json` archivo.

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

Paso 5: Obtenga el punto final del tema MQTT de destino

Utilice el tema MQTT de destino para configurar dónde publican sus temas los mensajes salientes, sustituyendo la funcionalidad que antes utilizaba. AWS IoT Events El punto final es exclusivo de su AWS cuenta y región.

Console

1. Abra la [consola de AWS IoT Core](#).
2. En la sección Connect del panel de navegación izquierdo, selecciona Configuración de dominio.
3. Elija la configuración del dominio IoT:DATA-ATS para abrir la página de detalles de la configuración.
4. Copie el valor del nombre de dominio. Este valor es el punto final. Guarde el valor del punto final, ya que lo necesitará en pasos posteriores.

AWS CLI

Ejecuta el siguiente comando para obtener el AWS IoT Core punto final para publicar los mensajes salientes de tu cuenta.

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

Paso 6: Crear una tabla de Amazon DynamoDB

Una tabla de Amazon DynamoDB reemplaza la funcionalidad de administración de estados AWS IoT Events de, lo que proporciona una forma escalable y flexible de conservar y administrar el estado de sus dispositivos y la lógica del modelo de detector en la nueva arquitectura de soluciones.

Console

Cree una tabla de Amazon DynamoDB para conservar el estado de los modelos de detectores. Para obtener más información, consulte [Crear una tabla en DynamoDB en](#) la Guía para desarrolladores de Amazon DynamoDB.

Utilice lo siguiente para ver los detalles de la tabla:

- En Nombre de tabla, introduzca el nombre de tabla que desee.
- En Clave de partición, introduce tu propio ID de instancia.
- Puede usar la configuración predeterminada para la configuración de la tabla

AWS CLI

Ejecute el siguiente comando para crear una tabla de DynamoDB.

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

Paso 7: Crear una AWS Lambda función (consola)

La función Lambda actúa como motor de procesamiento central y reemplaza la lógica de evaluación del modelo de detector de. AWS IoT Events En el ejemplo, nos integramos con otros AWS servicios para gestionar los datos entrantes, gestionar el estado y activar acciones en función de las reglas definidas.

Cree una función Lambda con NodeJS tiempo de ejecución. Utilice el siguiente fragmento de código para sustituir las constantes codificadas de forma rígida:

1. Abra la [AWS Lambda console](#).
2. Elija Crear función.
3. Introduzca un nombre para el nombre de la función.
4. Seleccione Nodejs 22.x como motor de ejecución.
5. En el menú desplegable Cambiar la función de ejecución predeterminada, elija Usar la función existente y, a continuación, seleccione la función de IAM que creó en los pasos anteriores.
6. Elija Crear función.
7. Pegue el siguiente fragmento de código después de reemplazar las constantes codificadas de forma rígida.
8. Una vez creada la función, en la pestaña Código, pega el siguiente ejemplo de código y reemplaza el **your-destination-endpoint** punto final por el tuyo.

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

```
import { SchedulerClient, CreateScheduleCommand, DeleteScheduleCommand } from "@aws-
sdk/client-scheduler"; // ES Modules import

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

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

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

  const processedRecords = [];

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

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

      const output = await handleDecodedData(decodedData);

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

```
        processedRecords.push(processedData);

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

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

// Helper function to handle decoded data
async function handleDecodedData(payload) {
    try {
        // Parse the decoded data
        const parsedData = JSON.parse(payload);

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

        await iotEvents.process(instanceId, inputData)

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

```

        throw error;
    }
}

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

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

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

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

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

```

```

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

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

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

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

    const scheduleName = `your-schedule-name-${instanceId}-schedule`;
    const scheduleParams = {
        Name: scheduleName,
        FlexibleTimeWindow: {
            Mode: 'OFF'
        },
        ScheduleExpression: `rate(${frequencyInMinutes} minutes)`,
        Target: {
            Arn: "arn:aws::kinesis:your-region:your-account-id:stream/your-kinesis-
stream-name",
            RoleArn: "arn:aws::iam:your-account-id:role/service-role/your-iam-
role",
            Input: base64Payload,
            KinesisParameters: {
                PartitionKey: instanceId,
            },
            RetryPolicy: {
                MaximumRetryAttempts: 3
            }
        },
    };

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

```

```

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

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

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

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

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

class IoTEvents {

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

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

    async process(instanceId, input) {

```

```

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

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

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

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

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

class IoTEventsState {
  constructor() {
    this.eventsList = []
  }

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

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

    return currentState;
  }
}

```

```

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

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

let processNormalEvent = new Event(
  (currentState) => currentState.input('temperature') > 70,
  async (currentState) => {
    currentState.changeState('alarm');
    await currentState.executeAction('MQTT', `{"state": "alarm detected, timer
started" }`);
    await currentState.setTimer(currentState.instanceId, 5, {

```

```
        "instanceId": currentState.instanceId,  
        "payload": "{\"currentState\": \"alarm\", \"source\": \"timer\"}"  
    });  
}  
);  
const iotEvents = new IoTEvents('normal');  
iotEvents  
    .state('normal')  
    .events(  
        [  
            processNormalEvent  
        ]  
    );  
iotEvents  
    .state('alarm')  
    .events([  
        processAlarmStateEvent,  
        processTimerEvent  
    ]  
    );
```

Paso 8: Añadir un activador de Amazon Kinesis Data Streams

Añada un activador de Kinesis Data Streams a la función Lambda mediante o. AWS Management Console AWS CLI

Al añadir un activador de Kinesis Data Streams a la función Lambda, se establece la conexión entre el proceso de ingesta de datos y la lógica de procesamiento, lo que le permite evaluar automáticamente los flujos de datos de IoT entrantes y reaccionar ante los eventos en tiempo real, de forma similar a como se procesa las entradas. AWS IoT Events

Console

Para obtener más información, consulte [Crear un mapeo de origen de eventos para invocar una función Lambda](#) en AWS Lambda la Guía para desarrolladores.

Utilice lo siguiente para obtener los detalles del mapeo de la fuente de eventos:

- En Nombre de función, introduzca el nombre de lambda utilizado en [Paso 7: Crear una AWS Lambda función \(consola\)](#).
- Para Consumer (opcional), introduzca el ARN de la transmisión de Kinesis.
- En Tamaño del lote, ingrese **10**.

AWS CLI

Ejecute el siguiente comando para crear el activador de la función Lambda.

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

Paso 9: Pruebe la funcionalidad de entrada y salida de datos ()AWS CLI

Publique una carga útil en el tema MQTT en función de lo que haya definido en su modelo de detector. El siguiente es un ejemplo de carga útil del tema `your-topic-name` MQTT para probar una implementación.

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

Debería ver un mensaje de MQTT publicado en un tema con el siguiente contenido (o similar):

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

Procedimiento de migración de AWS IoT SiteWise alarmas en AWS IoT Events

En esta sección, se describen soluciones alternativas que ofrecen una funcionalidad de alarma similar a la que se migra AWS IoT Events.

En el AWS IoT SiteWise caso de las propiedades que utilizan AWS IoT Events alarmas, puede migrar a una solución que utilice CloudWatch alarmas. Este enfoque proporciona capacidades de monitoreo sólidas con funciones establecidas SLAs y adicionales, como la detección de anomalías y las alarmas agrupadas.

Comparación de arquitecturas

La configuración de AWS IoT Events alarmas actual para AWS IoT SiteWise las propiedades debe crearse `AssetModelCompositeModels` en el modelo de activos, tal y como se describe en la [sección Definir alarmas externas](#) de la Guía del AWS IoT SiteWise usuario. AWS IoT SiteWise Por lo general, las modificaciones de la nueva solución se gestionan a través de la AWS IoT Events consola.

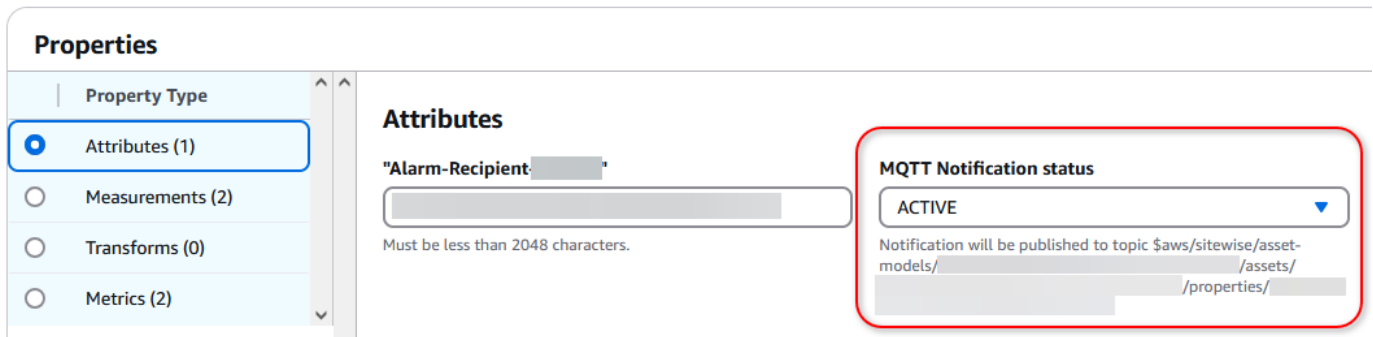
La nueva solución proporciona administración de alarmas mediante el aprovechamiento de CloudWatch las alarmas. Este enfoque utiliza AWS IoT SiteWise notificaciones para publicar puntos de datos de propiedades en temas de AWS IoT Core MQTT, que luego son procesados por una función Lambda. La función transforma estas notificaciones en CloudWatch métricas, lo que permite monitorizar las alarmas a través CloudWatch del marco de alarmas que incorpora.

Finalidad	Solución	Diferencias
Fuente de datos: datos de propiedades de AWS IoT SiteWise	AWS IoT SiteWise Notificaciones MQTT	Sustituye la integración directa de IoT Events con notificaciones MQTT de las propiedades AWS IoT SiteWise
Procesamiento de datos: transform a los datos de las propiedades	Función de Lambda	Procesa las notificaciones de AWS IoT SiteWise propiedades y las convierte en CloudWatch métricas
Evaluación de alarmas: monitorea las métricas y activa las alarmas	CloudWatch Alarmas Amazon	Sustituye las AWS IoT Events CloudWatch alarmas por alarmas y ofrece funciones adicionales, como la detección de anomalías
Integración: conexión con AWS IoT SiteWise	AWS IoT SiteWise alarmas externas	Capacidad opcional para volver a importar CloudWatch las alarmas AWS IoT SiteWise como alarmas externas

Paso 1: Habilite las notificaciones MQTT en la propiedad del activo

Si utiliza AWS IoT Events integraciones para las AWS IoT SiteWise alarmas, puede activar las notificaciones MQTT para cada propiedad que desee supervisar.

1. Siga el AWS IoT SiteWise procedimiento de [configuración de las alarmas de los activos](#) hasta llegar a editar las propiedades del modelo de activos.
2. Para cada propiedad que vaya a migrar, cambie el estado de la notificación MQTT a ACTIVO.



3. Anote la ruta temática en la que se publica la alarma para cada atributo de alarma modificado.

Para obtener más información, consulte los siguientes recursos de documentación:

- [Conozca las propiedades de los activos en los temas de MQTT](#) de la Guía del AWS IoT SiteWise usuario.
- [Los temas de MQTT](#) de la Guía para AWS IoT desarrolladores.

Paso 2: Crear una función de AWS Lambda

Cree una función Lambda para leer la matriz TQV publicada en el tema MQTT y publique valores individuales en ella. CloudWatch Usaremos esta función Lambda como acción de destino para activarla en AWS IoT Core Message Rules.

1. Abra la [AWS Lambda console](#).
2. Elija Crear función.
3. Introduzca un nombre para el nombre de la función.
4. Seleccione Nodejs 22.x como motor de ejecución.
5. En el menú desplegable Cambiar la función de ejecución predeterminada, elija Usar la función existente y, a continuación, seleccione la función de IAM que creó en los pasos anteriores.

 Note

Este procedimiento supone que ya ha migrado su modelo de detector. Si no tiene una función de IAM, consulte [???](#).

6. Elija Crear función.
7. Pegue el siguiente fragmento de código después de reemplazar las constantes codificadas de forma rígida.

```
import json
import boto3
from datetime import datetime

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

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

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

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

```

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

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

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

```

Paso 3: Crea AWS IoT Core una regla de enrutamiento de mensajes

- Siga el [tutorial: Procedimiento para volver a publicar un mensaje MQTT](#) introduciendo la siguiente información cuando se le solicite:
 - a. Nombre la regla de enrutamiento de mensajes. SiteWiseToCloudwatchAlarms
 - b. Para la consulta, puede usar lo siguiente:

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

- c. En Acciones de regla, seleccione la acción Lambda desde AWS IoT SiteWise la que enviar los datos generados a. CloudWatch Por ejemplo:

Rule actions [Info](#)

Select one or more actions to happen when the above rule is matched by an inbound message. Actions define additional activities that occur when messages arrive, like storing them in a database, invoking cloud functions, or sending notifications. You can add up to 10 actions.

Action 1

▼ **Lambda** [Remove](#)

Send a message to a Lambda function

Lambda function [Info](#)

ListenForSiteWiseUpdates [View](#) [Create a Lambda function](#)

Lambda function version

\$LATEST [View](#)

[Add rule action](#)

Paso 4: Ver CloudWatch las métricas

A medida que ingiere datos a AWS IoT SiteWise la propiedad seleccionada anteriormente [Paso 1: Habilite las notificaciones MQTT en la propiedad del activo](#), los dirige a la función Lambda que creamos en. [Paso 2: Crear una función de AWS Lambda](#) En este paso, puedes comprobar a qué Lambda envía tus métricas. CloudWatch

1. Abra la [CloudWatch de AWS Management Console](#).
2. En el menú de navegación de la izquierda, selecciona Métricas y, a continuación, Todas las métricas.
3. Elige la URL de una alarma para abrirla.
4. En la pestaña Fuente, el CloudWatch resultado tiene un aspecto similar al de este ejemplo. La información de esta fuente confirma que se están introduciendo los datos de las métricas CloudWatch.

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

Paso 5: Crear CloudWatch alarmas

Sigue el procedimiento [Crear una CloudWatch alarma basada en un umbral estático](#) de la Guía del CloudWatch usuario de Amazon para crear alarmas para cada métrica relevante.

Note

Existen muchas opciones de configuración de alarmas en Amazon. CloudWatch Para obtener más información sobre CloudWatch las alarmas, consulta [Uso de CloudWatch alarmas de Amazon](#) en la Guía del CloudWatch usuario de Amazon.

Paso 6: (opcional) importe la CloudWatch alarma a AWS IoT SiteWise

Puede configurar CloudWatch las alarmas para devolver los datos AWS IoT SiteWise mediante acciones de CloudWatch alarma y Lambda. Esta integración le permite ver los estados y las propiedades de las alarmas en el portal SiteWise Monitor.

1. Configure la alarma externa como una propiedad en un modelo de activos. Para obtener más información, consulte [Definir alarmas externas AWS IoT SiteWise en](#) la Guía del AWS IoT SiteWise usuario.
2. Cree una función Lambda que utilice la [BatchPutAssetPropertyValueAPI](#) que se encuentra en la Guía del AWS IoT SiteWise usuario para enviar datos de alarma a. AWS IoT SiteWise
3. Configure acciones de CloudWatch alarma para invocar la función Lambda cuando cambien los estados de alarma. Para obtener más información, consulta la sección [Acciones de alarma](#) en la Guía del CloudWatch usuario de Amazon.

Con AWS IoT Events figuración

Esta sección proporciona una guía de configuración AWS IoT Events, que incluye la creación de una AWS cuenta, la configuración de los permisos necesarios y el establecimiento de funciones para administrar el acceso a los recursos.

Temas

- [Configurar un Cuenta de AWS](#)
- [Configuración de permisos para AWS IoT Events](#)

Configurar un Cuenta de AWS

Inscríbase en un Cuenta de AWS

Si no tiene una Cuenta de AWS, complete los siguientes pasos para crearlo.

Para suscribirte a una Cuenta de AWS

1. Abrir <https://portal.aws.amazon.com/billing/registro>.
2. Siga las instrucciones que se le indiquen.

Parte del procedimiento de registro consiste en recibir una llamada telefónica o mensaje de texto e indicar un código de verificación en el teclado del teléfono.

Cuando te registras en una Cuenta de AWS, Usuario raíz de la cuenta de AWS se crea uno. El usuario raíz tendrá acceso a todos los Servicios de AWS y recursos de esa cuenta. Como práctica recomendada de seguridad, asigne acceso administrativo a un usuario y utilice únicamente el usuario raíz para realizar [tareas que requieren acceso de usuario raíz](#).

AWS te envía un correo electrónico de confirmación una vez finalizado el proceso de registro. En cualquier momento, puede ver la actividad de su cuenta actual y administrarla accediendo a <https://aws.amazon.com/> y seleccionando Mi cuenta.

Creación de un usuario con acceso administrativo

Después de crear un usuario administrativo Cuenta de AWS, asegúrelo Usuario raíz de la cuenta de AWS AWS IAM Identity Center, habilite y cree un usuario administrativo para no usar el usuario root en las tareas diarias.

Proteja su Usuario raíz de la cuenta de AWS

1. Inicie sesión [AWS Management Console](#) como propietario de la cuenta seleccionando el usuario root e introduciendo su dirección de Cuenta de AWS correo electrónico. En la siguiente página, escriba su contraseña.

Para obtener ayuda para iniciar sesión con el usuario raíz, consulte [Iniciar sesión como usuario raíz](#) en la Guía del usuario de AWS Sign-In .

2. Active la autenticación multifactor (MFA) para el usuario raíz.

Para obtener instrucciones, consulte [Habilitar un dispositivo MFA virtual para el usuario Cuenta de AWS raíz \(consola\)](#) en la Guía del usuario de IAM.

Creación de un usuario con acceso administrativo

1. Activar IAM Identity Center.

Consulte las instrucciones en [Activar AWS IAM Identity Center](#) en la Guía del usuario de AWS IAM Identity Center .

2. En IAM Identity Center, conceda acceso administrativo a un usuario.

Para ver un tutorial sobre su uso Directorio de IAM Identity Center como fuente de identidad, consulte [Configurar el acceso de los usuarios con la configuración predeterminada Directorio de IAM Identity Center en la](#) Guía del AWS IAM Identity Center usuario.

Inicio de sesión como usuario con acceso de administrador

- Para iniciar sesión con el usuario de IAM Identity Center, use la URL de inicio de sesión que se envió a la dirección de correo electrónico cuando creó el usuario de IAM Identity Center.

Para obtener ayuda para iniciar sesión con un usuario del Centro de identidades de IAM, consulte [Iniciar sesión en el portal de AWS acceso](#) en la Guía del AWS Sign-In usuario.

Concesión de acceso a usuarios adicionales

1. En IAM Identity Center, cree un conjunto de permisos que siga la práctica recomendada de aplicar permisos de privilegios mínimos.

Para conocer las instrucciones, consulte [Create a permission set](#) en la Guía del usuario de AWS IAM Identity Center .

2. Asigne usuarios a un grupo y, a continuación, asigne el acceso de inicio de sesión único al grupo.

Para conocer las instrucciones, consulte [Add groups](#) en la Guía del usuario de AWS IAM Identity Center .

Configuración de permisos para AWS IoT Events

La implementación de los permisos adecuados es importante para un uso seguro y eficaz de AWS IoT Events. En esta sección se describen los permisos necesarios para utilizar algunas funciones de AWS IoT Events. Puede usar AWS CLI los comandos o la consola AWS Identity and Access Management (IAM) para crear roles y políticas de permisos asociadas para acceder a los recursos o realizar determinadas funciones. AWS IoT Events

La [guía del usuario de IAM](#) contiene información más detallada sobre cómo controlar de forma segura los permisos de acceso AWS a los recursos. Para obtener información específica sobre AWS IoT Events, consulte [Acciones, recursos y claves de condición para AWS IoT Events](#).

Si quieres usar la consola de IAM para crear y gestionar funciones y permisos, consulta el [tutorial de IAM: Delega el acceso entre AWS cuentas mediante funciones de IAM](#).

Note

Las claves pueden tener entre 1 y 128 caracteres e incluir:

- letras (a-z) en mayúsculas y minúsculas
- números (0-9)
- caracteres especiales (-, _ o :).

Permisos de acción para AWS IoT Events

AWS IoT Events le permite activar acciones que utilizan otros AWS servicios. Para ello, debe conceder AWS IoT Events permiso para realizar estas acciones en su nombre. Esta sección contiene una lista de las acciones y una política de ejemplo que concede permiso para realizar todas estas acciones en sus recursos. Cambie las *account-id* referencias *region* y según sea necesario. De ser posible, también debería cambiar los comodines (*) para hacer referencia a los recursos específicos a los que se accederá. Puede utilizar la consola de IAM para conceder permiso AWS IoT Events para enviar una alerta de Amazon SNS que haya definido.

AWS IoT Events admite las siguientes acciones que le permiten usar un temporizador o configurar una variable:

- [setTimer](#) para crear un temporizador.
- [resetTimer](#) para restablecer el temporizador.
- [clearTimer](#) para eliminar el temporizador.
- [setVariable](#) para crear una variable.

AWS IoT Events admite las siguientes acciones que le permiten trabajar con AWS los servicios:

- [iotTopicPublish](#) para publicar un mensaje en un tema de MQTT.
- [iotEvents](#) para enviar datos a AWS IoT Events como valor de entrada.
- [iotSiteWise](#) para enviar datos a una propiedad de recurso en AWS IoT SiteWise.
- [dynamoDB](#) para enviar datos a una tabla de Amazon DynamoDB.
- [dynamoDBv2](#) para enviar datos a una tabla de Amazon DynamoDB.
- [firehose](#) para enviar datos a una transmisión de Amazon Data Firehose.
- [lambda](#) para invocar una función de AWS Lambda .
- [sns](#) para enviar datos como notificación de inserción.
- [sqs](#) para enviar datos a una cola de Amazon SQS.

Example Política

```
{
  "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>:*"
  }
]

```

```
}
```

Proteger los datos de entrada en AWS IoT Events

Es importante tener en cuenta quién puede conceder acceso a los datos de entrada para su uso en un modelo de detector. Si tiene un usuario o entidad cuyos permisos generales desea restringir, pero que tenga permiso para crear o actualizar un modelo de detector, también debe conceder permiso para que ese usuario o entidad actualice la ruta de entrada. Esto significa que además de conceder permiso para `iotevents:CreateDetectorModel` y `iotevents:UpdateDetectorModel`, también debe conceder permiso para `iotevents:UpdateInputRouting`.

Example

La siguiente política añade permiso para `iotevents:UpdateInputRouting`.

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

Puede especificar una lista de entradas de Amazon Resource Names (ARNs) en lugar del comodín `*` para el "Resource" para limitar este permiso a entradas específicas. Esto le permite restringir el acceso a los datos de entrada que consumen los modelos de detector creados o actualizados por el usuario o la entidad.

Política de funciones de CloudWatch registro de Amazon para AWS IoT Events

Los siguientes documentos de política proporcionan la política de roles y la política de confianza que AWS IoT Events permiten enviar registros CloudWatch en su nombre.

Política de roles:

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

Política de confianza:

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

También necesita una política de permisos de IAM asociada al usuario que le permita pasar roles, de la siguiente manera. Para obtener más información, consulte [Otorgar permisos a un usuario para transferir un rol a un AWS servicio](#) en la Guía del usuario de IAM.

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

Puede usar el siguiente comando para colocar la política de recursos para CloudWatch los registros. Esto permite AWS IoT Events colocar los eventos de registro en las CloudWatch transmisiones.

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

Utilice el siguiente comando para colocar las opciones de registro. Sustituya el `roleArn` por el rol de registro que haya creado.

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

Política de funciones de mensajería de Amazon SNS para AWS IoT Events

La integración AWS IoT Events con Amazon SNS requiere una gestión cuidadosa de los permisos para una entrega de notificaciones segura y eficiente. Esta guía explica el proceso de configuración de las funciones y políticas de IAM AWS IoT Events para permitir la publicación de mensajes en los temas de Amazon SNS.

Los siguientes documentos de política proporcionan la política de roles y la política de confianza que permiten a AWS IoT Events enviar mensajes SNS.

Política de roles:

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

Política de confianza:

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

Primeros pasos con la AWS IoT Events consola

En esta sección le mostramos cómo crear una entrada y un modelo de detector utilizando la [consola de AWS IoT Events](#). Se modelan dos estados de un motor: un estado normal y un estado de sobrepresión. Cuando la presión medida en el motor supera un determinado umbral, el modelo pasa del estado normal al estado de sobrepresión. A continuación, envía un mensaje de Amazon SNS para alertar a un técnico sobre la condición. Cuando la presión vuelve a caer por debajo del umbral durante tres lecturas de presión consecutivas, el modelo vuelve al estado normal y envía otro mensaje de Amazon SNS como confirmación.

Comprobamos que haya tres lecturas consecutivas por debajo del umbral de presión para eliminar posibles tartamudeos de mensajes de estado de sobrepresión o normal por si hubiera una fase de recuperación no lineal o una lectura de presión anómala.

En la consola también puede encontrar varias plantillas prefabricadas de modelos de detectores que puede personalizar. También puede usar la consola para importar modelos de detectores que otros hayan escrito o exportar sus modelos de detector y usarlos en diferentes AWS regiones. Si importa un modelo de detector, asegúrese de crear las entradas necesarias o de volver a crearlas para la nueva región y actualice cualquier función ARNs utilizada.

Usa la AWS IoT Events consola para obtener información sobre lo siguiente.

Definir entradas

Para supervisar sus dispositivos y procesos, deben tener una forma de transferir datos de telemetría a AWS IoT Events. Esto se hace enviando mensajes como entradas a AWS IoT Events. Puede hacer esto de varias formas:

- Utilice la operación [BatchPutMessage](#).
- En AWS IoT Core, escribe una regla de [AWS IoT Events acción](#) para el motor de AWS IoT reglas al que reenvía los datos de tus mensajes. AWS IoT Events Debe identificar la entrada por su nombre.
- En AWS IoT Analytics, utilice la [CreateDataset](#) operación para crear un conjunto de datos con `contentDeliveryRules`. Estas reglas especifican la AWS IoT Events entrada a la que se envía automáticamente el contenido del conjunto de datos.

Para que sus dispositivos puedan enviar datos de este modo, debe definir una o más entradas. Para ello, asigne un nombre a cada entrada y especifique qué campos de los datos del mensaje entrante supervisa la entrada.

Crear un modelo de detector

Defina un modelo de detector (un modelo de su equipo o proceso) utilizando estados. Para cada estado, defina una lógica condicional (booleana) que evalúe las entradas para detectar eventos significativos. Cuando el modelo de detector detecta un evento, puede cambiar el estado o iniciar acciones personalizadas o predefinidas mediante otros AWS servicios. Puede definir eventos adicionales que inicien acciones al entrar o salir de un estado y, opcionalmente, al cumplirse una condición.

En este tutorial, usted envía un mensaje de Amazon SNS como acción cuando el modelo entra o sale de un determinado estado.

Cómo supervisar un dispositivo o proceso

Si supervisa varios dispositivos o procesos, especifique un campo en cada entrada que identifique el dispositivo o proceso en particular del que procede la entrada. Consulte el campo `key` en `CreateDetectorModel`. Cuando el campo de entrada identificado por `key` reconoce un nuevo valor, se identifica un nuevo dispositivo y se crea un detector. Cada detector es una instancia del modelo de detector. El nuevo detector sigue respondiendo a las entradas procedentes de ese dispositivo hasta que actualice o elimine su modelo de detector.

Si supervisa un único proceso (incluso con varios dispositivos o subprocesos enviando entradas), no especifica un campo `key` de identificación único. En este caso, el modelo crea un único detector (instancia) cuando llega la primera entrada.

Cómo enviar mensajes como entradas a su modelo de detector

Existen varias formas de enviar un mensaje desde un dispositivo o proceso como entrada a un detector de AWS IoT Events que no requieren que realice un formateo adicional del mensaje. En este tutorial, utilizará la AWS IoT consola para escribir una regla de [AWS IoT Events acción](#) para el motor de AWS IoT reglas al que reenvía los datos de sus mensajes. AWS IoT Events

Para ello, identifique la entrada por su nombre y siga utilizando la AWS IoT consola para generar mensajes que se reenvíen como entradas. AWS IoT Events

Note

En este tutorial utiliza la consola para crear los mismos `input` y `detector model` que se muestran en el ejemplo en [Tutoriales para casos de AWS IoT Events uso](#). Puede utilizar este ejemplo JSON como ayuda para seguir el tutorial.

Temas

- [Requisitos previos para empezar AWS IoT Events](#)
- [Cree una entrada para los modelos en AWS IoT Events](#)
- [Cree un modelo de detector en AWS IoT Events](#)
- [Envíe las entradas para probar el modelo de detector en AWS IoT Events](#)

Requisitos previos para empezar AWS IoT Events

Si no tienes una AWS cuenta, crea una.

1. Sigue estos pasos [Con AWS IoT Events figuración](#) para asegurarte de que la cuenta y los permisos estén configurados correctamente.
2. Cree dos temas de Amazon Simple Notification Service (Amazon SNS).

En este tutorial (y en el ejemplo correspondiente) se supone que ha creado dos temas de Amazon SNS. Estos temas se muestran como: `arn:aws:sns:us-east-1:123456789012:underPressureAction` y `arn:aws:sns:us-east-1:123456789012:pressureClearedAction`. ARNs Sustituya estos valores por los temas ARNs de Amazon SNS que cree. Para obtener más información, consulte la [Guía para desarrolladores de Amazon Simple Notification Service](#).

Como alternativa a la publicación de alertas para temas de Amazon SNS, puede hacer que los detectores envíen mensajes MQTT con un tema que usted especifique. Con esta opción, puede comprobar que su modelo de detector está creando instancias y que esas instancias envían alertas mediante la consola AWS IoT Core para suscribirse a los mensajes enviados a esos temas de MQTT y supervisarlos. También puede definir el nombre del tema de MQTT dinámicamente en tiempo de ejecución utilizando una entrada o variable creada en el modelo de detector.

3. Elija una Región de AWS que sea compatible AWS IoT Events. Para obtener más información, consulte [AWS IoT Events](#) en la Referencia general de AWS. Para [obtener ayuda, consulte Cómo empezar con un servicio AWS Management Console en la](#) sección Cómo empezar con AWS Management Console.

Cree una entrada para los modelos en AWS IoT Events

Al construir las entradas para sus modelos, le recomendamos que reúna archivos que contengan cargas de mensajes de muestra que sus dispositivos o procesos envíen para informar de su estado. Disponer de estos archivos le ayudará a definir las entradas necesarias.

Puede crear una entrada a través de varios métodos, que se describen en esta sección.

Cree un archivo de entrada JSON

1. Para empezar, cree un archivo llamado `input.json` en su sistema de archivos local con el siguiente contenido:

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

2. Ahora que tiene este archivo `input.json` de inicio, puede crear una entrada. Hay dos formas de crear una entrada. Puede crear una entrada mediante el panel de navegación de la [AWS IoT Events consola](#). O bien, puede crear una entrada dentro del modelo de detector después de crearla.

Cree y configure una entrada

Aprenda a crear una entrada para un modelo de alarma o un modelo de detector.

1. Inicie sesión en la [AWS IoT Events consola](#) o seleccione la opción Crear una AWS IoT Events cuenta nueva.
2. En la AWS IoT Events consola, en la esquina superior izquierda, selecciona y expande el panel de navegación.
3. En el panel de navegación izquierdo, seleccione Entradas.
4. En la esquina derecha de la consola, seleccione Crear entrada.
5. Proporcione un único InputName.
6. Opcional: introduce una descripción para tu entrada.

7. Para cargar un archivo JSON, selecciona el `input.json` archivo para el que has creado en la descripción general [Cree un archivo de entrada JSON](#). La opción Elegir atributos de entrada aparece junto con una lista de los atributos introducidos.
8. En Elegir atributos de entrada, seleccione los atributos que desee utilizar y luego Crear. En este ejemplo, seleccionamos `motorid` y `sensorData.pressure`.
9. Opcional: añada las etiquetas pertinentes a la entrada.

 Note

También puede crear entradas adicionales dentro del modelo de detector de la [AWS IoT Events consola](#). Para obtener más información, consulte [Cree una entrada dentro del modelo de detector en AWS IoT Events](#).

Cree una entrada dentro del modelo de detector en AWS IoT Events

Las entradas del detector AWS IoT Events sirven de puente entre las fuentes de datos y los modelos de detectores. Las entradas del detector proporcionan los datos sin procesar que impulsan las capacidades de automatización y detección de eventos de AWS IoT Events. Aprenda a configurar las entradas de los detectores para ayudar a sus modelos a responder con precisión a los eventos y condiciones del mundo real en su ecosistema de IoT.

En esta sección se muestra cómo definir una entrada para que un modelo de detector reciba datos o mensajes de telemetría.

Para definir una entrada para un modelo de detector

1. Abra la [consola de AWS IoT Events](#).
2. En la AWS IoT Events consola, elija Crear modelo de detector.
3. Elija Crear nuevo.
4. Elija Create input (Crear entrada).
5. Para la entrada, introduzca una `InputName`, una descripción opcional y elija Cargar archivo. En el cuadro de diálogo que aparece, seleccione el `input.json` archivo para el que creó en la vista general [Cree un archivo de entrada JSON](#).
6. En Elegir atributos de entrada, seleccione los atributos que desee utilizar y luego Crear. En este ejemplo, seleccionamos `MotorID` y `SensorData.PRESSURE`.

Cree un modelo de detector en AWS IoT Events

En este tema, define un modelo de detector (un modelo de su equipo o proceso) utilizando estados.

Para cada estado, debe definir una lógica condicional (booleana) que evalúe las entradas para detectar un evento significativo. Al detectar un evento, este cambia el estado y puede iniciar acciones adicionales. Estos eventos se conocen como eventos de transición.

En sus estados, también define eventos que pueden ejecutar acciones cada vez que el detector entra o sale de ese estado o cuando se recibe una entrada (se conocen como eventos `OnEnter`, `OnExit` y `OnInput`). Las acciones se ejecutan solo si la lógica condicional del evento da como resultado `true`.

Para crear un modelo de detector

1. Se ha creado el primer estado del detector para usted. Para modificarlo, seleccione el círculo con la etiqueta `State_1` en el espacio de edición principal.
2. En el panel Estado, introduzca el nombre del estado y `OnEnter` elija Añadir evento.
3. En la página Añadir `OnEnter` evento, introduzca el nombre del evento y la condición del evento. En este ejemplo, introduzca `true` para indicar que el evento siempre se inicia al introducir el estado.
4. En Acciones del evento, seleccione Añadir acción.
5. En Acciones del evento, realice lo siguiente:
 - a. Seleccione Establecer variable
 - b. En Operación de la variable, seleccione Asignar valor.
 - c. En Nombre de la variable, introduzca el nombre de la variable que va a establecer.
 - d. En Valor de la variable, introduzca el valor `0` (cero).
6. Seleccione Guardar.

Una variable, como la que ha definido, se puede establecer (darle un valor) en cualquier evento del modelo de detector. Solo se puede hacer referencia al valor de la variable (por ejemplo, en la lógica condicional de un evento) después de que el detector haya alcanzado un estado y ejecutado una acción donde esté definido o establecido.

7. En el panel Estado, seleccione la X situada junto a Estado para volver a la Paleta de modelos de detectores.

8. Para crear un segundo estado de detector, en la Paleta de modelos de detectores, seleccione Estado y arrástrelo al espacio de edición principal. Esto crea un estado titulado `untitled_state_1`.
9. Haga una pausa en el primer estado (Normal). Aparece una flecha en la circunferencia de estado.
10. Pulse y arrastre la flecha del primer estado al segundo estado. Aparece una línea dirigida del primer estado al segundo estado (denominada Sin título).
11. Seleccione la línea Sin título. En el panel Evento de transición, introduzca el Nombre del evento y una Lógica de activación del evento.
12. En el panel Evento de transición, seleccione Añadir acción.
13. En el panel Añadir acciones de evento de transición, seleccione Añadir acción.
14. En Elegir una acción, seleccione Establecer variable.
 - a. En Operación de la variable, seleccione Asignar valor.
 - b. En Nombre de la variable, introduzca el nombre de la variable.
 - c. En Asignar valor, escriba el valor, como `$variable.pressureThresholdBreached + 3`:
 - d. Seleccione Guardar.
15. Seleccione el segundo estado `untitled_state_1`.
16. En el panel Estado, introduzca el Nombre del estado y en OnEnter, seleccione Añadir evento.
17. En la página Añadir OnEnter evento, introduzca el nombre del evento y la condición del evento. Seleccione Agregar acción.
18. En Seleccionar una acción, seleccione Enviar mensaje SNS.
 - a. En Tema de SNS, introduzca el ARN objetivo de su tema de Amazon SNS.
 - b. Seleccione Guardar.
19. Siga añadiendo los eventos indicados en el ejemplo.
 - a. Para OnInput, elija Agregar evento e introduzca y guarde la siguiente información del evento.

```
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. Para OnInput, elija Agregar evento e introduzca y guarde la siguiente información del evento.

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

- c. Para OnExit, elija Añadir evento e introduzca y guarde la siguiente información del evento con el ARN del tema de Amazon SNS que ha creado.

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

20. Haga una pausa en el segundo estado (Peligroso). Aparece una flecha en la circunferencia del estado
21. Pulse y arrastre la flecha del segundo estado al primer estado. Aparece una línea dirigida con la etiqueta Sin título.
22. Seleccione la línea Sin título y en el panel Evento de transición, introduzca el Nombre del evento y la Lógica de activación del evento utilizando la siguiente información.

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

Para obtener más información sobre por qué probamos el valor `$input` y el valor `$variable` en la lógica de activación, consulte en la entrada la disponibilidad de los valores de las variables en [AWS IoT Events restricciones y limitaciones del modelo de detector](#).

23. Seleccione el estado Inicio. Por defecto, este estado se generó cuando creó un modelo de detector). En el panel Inicio, seleccione el Estado de destino (por ejemplo, Normal).
24. A continuación, configure su modelo de detector para que escuche las entradas. En la esquina superior derecha, seleccione Publicar.
25. En la página Publicar modelo de detector, haga lo siguiente.
 - a. Introduzca el Nombre del modelo de detector, una Descripción y el nombre de un Rol. Este rol se crea para usted.
 - b. Seleccione Crear un detector para cada valor clave único. Para crear su propio Rol y utilizarlo, siga los pasos que se indican en [Configuración de permisos para AWS IoT Events](#) e introdúzcalo como Rol aquí.
26. En Clave de creación del detector, elija el nombre de uno de los atributos de la entrada que definió anteriormente. El atributo que elija como clave de creación del detector debe estar presente en cada entrada de mensaje y debe ser único para cada dispositivo que envíe mensajes. En este ejemplo se utiliza el atributo `motorid`.
27. Seleccione Guardar y publicar.

Note

El número de detectores únicos creados para un modelo de detector determinado se basa en los mensajes de entrada enviados. Cuando se crea un modelo de detector, se selecciona una clave a partir de los atributos de entrada. Esta clave determina qué instancia de detector se utilizará. Si la clave no se ha visto antes (para este modelo de detector), se crea una nueva instancia de detector. Si se ha visto antes, se utiliza la instancia de detector existente correspondiente a este valor clave.

Puede hacer una copia de seguridad de la definición de su modelo de detector (en JSON), recrear o actualizar el modelo de detector, o utilizarlo como plantilla para crear otro modelo de detector.

Puede hacerlo desde la consola o utilizando el siguiente comando CLI. De ser necesario, cambie el nombre del modelo de detector para que coincida con el que utilizó al publicarlo en el paso anterior.

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

Esto crea un archivo (`motorDetectorModel.json`) que tiene un contenido similar al siguiente.

```
{
  "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":
"pressureThresholdBreach",
                                "value": "0"
                            }
                        }
                    ],
                    "condition": "true"
                }
            ],
            "onExit": {
                "events": []
            }
        },
        {
            "onInput": {
                "transitionEvents": [
                    {
                        "eventName": "Back to Normal",
                        "actions": [],
                        "condition": "$variable.pressureThresholdBreach <= 1
&& $input.PressureInput.sensorData.pressure <= 70",
                        "nextState": "Normal"
                    }
                ],
                "events": [
                    {
                        "eventName": "Overpressurized",
                        "actions": [
                            {
                                "setVariable": {
                                    "variableName":
"pressureThresholdBreach",
                                    "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"
}
}
```

Envíe las entradas para probar el modelo de detector en AWS IoT Events

Hay varias formas de recibir datos de telemetría en AWS IoT Events (consulte [Acciones compatibles para recibir datos y activar acciones en AWS IoT Events](#)). En este tema se muestra cómo crear una AWS IoT regla en la AWS IoT consola que reenvíe los mensajes como entradas al detector. AWS IoT Events Puede utilizar el cliente MQTT de la AWS IoT consola para enviar mensajes de prueba. Puede usar este método para obtener datos de telemetría para saber AWS IoT Events cuándo sus dispositivos pueden enviar mensajes MQTT mediante el intermediario de mensajes. AWS IoT

Para enviar entradas para probar el modelo de detector

1. Abra la [consola de AWS IoT Core](#). En el panel de navegación izquierdo, en Administrar, seleccione Enrutamiento de mensajes y luego Reglas.
2. Seleccione Crear regla en la esquina superior derecha.
3. En la página Crear una regla, siga los pasos que se indican a continuación:
 1. Paso 1. Especificar las propiedades de la regla. Complete los siguientes campos:
 - Nombre de la regla. Escriba un nombre para su regla, como MyIoTEventsRule.

 Note

No utilice espacios.

- Descripción de la regla. Es opcional.
- Elija Next (Siguiente).

2. Paso 2. Configurar la instrucción SQL. Complete los siguientes campos:

- Versión de SQL. Seleccione la opción apropiada en la lista.
- Sentencia SQL. Escriba **SELECT *, topic(2) as motorid FROM 'motors/+/status'**.

Elija Next (Siguiente).

3. Paso 3. Añadir acciones de la regla. En la sección Acciones de la regla, complete lo siguiente:

- Acción 1. Seleccione IoT Events. Se muestran los siguientes campos:
 - a. Nombre de la entrada. Seleccione la opción apropiada en la lista. Si la entrada no aparece, seleccione Actualizar.

Para crear una entrada nueva, seleccione Crear entrada de IoT Events. Complete los siguientes campos:

- Nombre de la entrada. Escriba `PressureInput`.
- Descripción. Es opcional.
- Cargar un archivo JSON. Cargue una copia de su archivo JSON. Si no tuviera un archivo, hay un enlace a un archivo de muestra en esta pantalla. El código incluye:

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

- Elegir atributos de entrada. Seleccione las opciones apropiadas.
- Etiquetas. Es opcional.

Seleccione Crear.

Vuelva a la pantalla Crear regla y actualice el campo Nombre de la entrada. Seleccione la entrada que acaba de crear.

- b. Modo Lote. Es opcional. Si la carga es una matriz de mensajes, seleccione esta opción.
- c. ID de mensaje. Esto es opcional, pero recomendable.
- d. Rol de IAM. Seleccione el rol apropiado en la lista. Si el rol no figura en la lista, seleccione Crear nuevo rol.

Escriba el Nombre del rol y seleccione Crear.

Para añadir otra regla, seleccione Añadir regla.

- Acción de error. Esta sección es opcional. Para añadir una acción, seleccione Añadir acción de error y seleccione la acción apropiada en la lista.

Complete los campos que aparezcan.

- Elija Next (Siguiente).

4. Paso 4. Revisar y crear. Revise la información en pantalla y seleccione Crear.

4. En el panel de navegación izquierdo, en Prueba, seleccione Cliente de prueba MQTT.

5. Elija Publicar en un tema. Complete los siguientes campos:

- Nombre del tema. Introduzca un nombre para identificar el mensaje, por ejemplo, `motors/Fulton-A32/status`.
- Carga útil del mensaje. Introduzca lo siguiente:

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

 Note

Cambie el `messageId` cada vez que publique un nuevo mensaje.

6. Para Publicar, mantenga el tema igual, pero cambie la `"pressure"` en la carga por un valor superior al valor umbral que especificó en el modelo de detector (por ejemplo **85**).

7. Seleccione Publicar.

La instancia de detector que ha creado genera y le envía un mensaje de Amazon SNS. Continúe enviando mensajes con lecturas de presión por encima o por debajo del umbral de presión (70 en este ejemplo) para ver el detector en funcionamiento.

En este ejemplo, debe enviar tres mensajes con lecturas de presión por debajo del umbral para volver al estado Normal y recibir un mensaje de Amazon SNS que indique que la condición de sobrepresión ha desaparecido. Una vez de nuevo en el estado Normal, un mensaje con una lectura de presión por encima del límite hace que el detector pase al estado Peligroso y envíe un mensaje de Amazon SNS indicando esa condición.

Ahora que ha creado un modelo de detector y entrada sencillos, pruebe lo siguiente.

- Consulte más ejemplos de modelos de detectores (plantillas) en la consola.
- Siga los pasos [Cree un AWS IoT Events detector para dos estados mediante CLI](#) que se indican para crear un modelo de entrada y detector utilizando el AWS CLI
- Conozca los detalles de las [Expresiones para filtrar, transformar y procesar datos de eventos](#) utilizadas en los eventos.
- Información sobre [Acciones compatibles para recibir datos y activar acciones en AWS IoT Events](#).
- Si algo no funciona, consulte [Solución de problemas AWS IoT Events](#).

Mejores prácticas para AWS IoT Events

Siga estas prácticas recomendadas para obtener el máximo beneficio de AWS IoT Events.

Temas

- [Habilite el CloudWatch registro de Amazon al desarrollar modelos AWS IoT Events de detectores](#)
- [Publique periódicamente para guardar su modelo de detector cuando trabaje en la AWS IoT Events consola](#)

Habilite el CloudWatch registro de Amazon al desarrollar modelos AWS IoT Events de detectores

Amazon CloudWatch monitorea tus AWS recursos y las aplicaciones en las que AWS ejecutas en tiempo real. De este CloudWatch modo, obtiene visibilidad en todo el sistema sobre el uso de los recursos, el rendimiento de las aplicaciones y el estado operativo. Cuando desarrolla o depura un modelo de AWS IoT Events detector, le CloudWatch ayuda a saber qué AWS IoT Events es lo que está haciendo y cualquier error que detecte.

Para habilitar CloudWatch

1. Si aún no lo ha hecho, siga los pasos que se indican [Configuración de permisos para AWS IoT Events](#) para crear un rol con una política adjunta que otorgue permiso para crear y administrar CloudWatch registros AWS IoT Events.
2. Vaya a la [consola de AWS IoT Events](#).
3. En el panel de navegación, seleccione Configuración.
4. En la página Configuración, seleccione Editar.
5. En la página Editar opciones de registro, en la sección Opciones de registro, haga lo siguiente:
 - a. En Nivel de verbosidad, seleccione una opción.
 - b. En Seleccionar rol, seleccione un rol con permisos suficientes para realizar las acciones de registro que haya elegido.
 - c. (Opcional) Si has elegido Depurar como nivel de verbosidad, puedes añadir objetivos de depuración de la siguiente manera:
 - i. En Objetivos de depuración, selecciona Añadir opción de modelo.

- ii. Introduzca un nombre de modelo de detector y (opcional) especifique KeyValues los modelos de detector y los detectores específicos (instancias) que desee registrar.

6. Elija Actualizar.

Sus opciones de registro se han actualizado correctamente.

Publique periódicamente para guardar su modelo de detector cuando trabaje en la AWS IoT Events consola

Al utilizar la AWS IoT Events consola, el trabajo en curso se guarda localmente en el navegador. Sin embargo, debe seleccionar Publicar para guardar su modelo de detector en AWS IoT Events. Después de publicar un modelo de detector, su trabajo publicado estará disponible en cualquier navegador que utilice para acceder a su cuenta.

Note

Si no publica su trabajo, no se guardará. Después de publicar un modelo de detector, ya no puede cambiar su nombre. Sin embargo, puede seguir modificando su definición.

Tutoriales para casos de AWS IoT Events uso

AWS IoT Events los tutoriales proporcionan una colección de procedimientos que abarcan varios aspectos AWS IoT Events, desde la configuración básica hasta casos de uso más específicos. Cada tutorial muestra ejemplos de escenarios prácticos, lo que le ayuda a desarrollar habilidades reales para crear modelos de detectores, configurar entradas, configurar acciones e integrarse con otros AWS servicios para crear potentes soluciones de IoT.

En este capítulo le mostramos cómo:

- Obtener ayuda para decidir qué estados incluir en su modelo de detector y determinar si necesita una instancia de detector o varias.
- Seguir un ejemplo que utilice la AWS CLI.
- Crear una entrada para recibir datos de telemetría de un dispositivo y un modelo de detector para supervisar el dispositivo que envía esos datos e informar de su estado.
- Revisar las restricciones y limitaciones de las entradas, los modelos de detector y el servicio de AWS IoT Events .
- Consultar un ejemplo más complejo de modelo de detector, con comentarios incluidos.

Temas

- [Uso AWS IoT Events para monitorear sus dispositivos de IoT](#)
- [Cree un AWS IoT Events detector para dos estados mediante CLI](#)
- [AWS IoT Events restricciones y limitaciones del modelo de detector](#)
- [Un ejemplo comentado: control de temperatura de HVAC con AWS IoT Events](#)

Uso AWS IoT Events para monitorear sus dispositivos de IoT

Puede usarlo AWS IoT Events para monitorear sus dispositivos o procesos y tomar medidas en función de eventos importantes. Para ello, siga estos pasos básicos:

Creación de entradas

Usted debe contar con un método para que sus dispositivos y procesos transmitan datos de telemetría a AWS IoT Events. Esto lo consigue enviando mensajes como entradas a AWS IoT Events. Puede enviar mensajes como entradas de varias maneras, para ello puede:

- Usa la [BatchPutMessage](#) operación.
- Definir una [rule-action de IoTEvents](#) para el [motor de reglas de AWS IoT Core](#). La rule-action reenvía los datos del mensaje desde su entrada a AWS IoT Events.
- En AWS IoT Analytics, utilice la [CreateDataset](#) operación para crear un conjunto de datos con `contentDeliveryRules`. Estas reglas especifican la AWS IoT Events entrada a la que se envía automáticamente el contenido del conjunto de datos.
- Defina una [acción de IoTEvents](#) en un evento `onExit` o `transitionEvents` modelo de AWS IoT Events `onInput` detector. La información sobre la instancia del modelo de detector y el evento que inició la acción se devuelven al sistema como una entrada con el nombre que usted especifique.

Antes de que sus dispositivos empiecen a enviar datos de esta forma, debe definir una o más entradas. Para ello, asigne un nombre a cada entrada y especifique qué campos de los datos del mensaje entrante monitorea la entrada. AWS IoT Events recibe su entrada, en forma de carga útil JSON, de muchas fuentes. Se puede actuar sobre cada entrada independiente o combinada con otras entradas para detectar eventos más complejos.

Crear un modelo de detector

Defina un modelo de detector (un modelo de su equipo o proceso) utilizando estados. Para cada estado, define una lógica condicional (booleana) que evalúa las entradas para detectar eventos significativos. Cuando se detecta un evento, puede cambiar el estado o iniciar acciones personalizadas o predefinidas mediante otros servicios. AWS Puede definir eventos adicionales que inicien acciones al entrar o salir de un estado y, opcionalmente, al cumplirse una condición.

En este tutorial, usted envía un mensaje de Amazon SNS como acción cuando el modelo entra o sale de un determinado estado.

Cómo supervisar un dispositivo o proceso

Al supervisar varios dispositivos o procesos, usted especifica un campo en cada entrada que identifica el dispositivo o proceso concreto del que procede la entrada. (Vea el campo `key` en `CreateDetectorModel`). Al identificarse un nuevo dispositivo (se ve un nuevo valor en el campo de entrada identificado por la `key`), se crea un detector. (Cada detector es una instancia del modelo de detector). A continuación, el nuevo detector sigue respondiendo a las entradas procedentes de ese dispositivo hasta que su modelo de detector se actualice o elimine.

Si supervisa un único proceso (incluso con varios dispositivos o subprocesos enviando entradas), usted no especifica un campo `key` de identificación único. En este caso, se crea un único detector (instancia) al llegar la primera entrada.

Cómo enviar mensajes como entradas a su modelo de detector

Existen varias formas de enviar un mensaje desde un dispositivo o proceso como entrada a un AWS IoT Events detector sin necesidad de formatear el mensaje de forma adicional. En este tutorial, utilizará la AWS IoT consola para escribir una regla de [AWS IoT Events acción](#) para el motor de AWS IoT Core reglas al que reenvía los datos de sus mensajes. AWS IoT Events Para ello, usted identifica la entrada por su nombre. A continuación, seguirá utilizando la AWS IoT consola para generar algunos mensajes que se reenviarán como entradas a AWS IoT Events.

¿Cómo puede saber qué estados necesita en un modelo de detector?

Para determinar qué estados debería tener su modelo de detector, primero decida qué medidas puede usted tomar. Por ejemplo, si su automóvil funciona con gasolina, usted mira el indicador de combustible al iniciar un viaje para ver si necesita repostar. Aquí tiene una acción: decirle al conductor “debe repostar”. Su modelo de detector necesita dos estados: “el coche no necesita combustible” y “el coche sí necesita combustible”. En general, querrá definir un estado para cada acción posible, más otro para cuando no se requiera ninguna acción. Esto funciona incluso si la acción en sí misma es más complicada. Por ejemplo, puede que quiera buscar e incluir información sobre dónde encontrar la gasolinera más cercana o el precio más barato, pero esto lo hace cuando envía el mensaje “debe repostar”.

Para decidir en qué estado debe entrar a continuación, se fijará en las entradas. Las entradas contienen la información que necesita para decidir en qué estado debería estar. Para crear una entrada, usted selecciona uno o varios campos en un mensaje enviado por su dispositivo o proceso que le ayuden a decidir. En este ejemplo, necesita una entrada que le indique el nivel actual de combustible (“porcentaje de lleno”). Es posible que su coche le envíe varios mensajes diferentes, cada uno con varios campos distintos. Para crear esta entrada, debe seleccionar el mensaje y el campo que informa del nivel actual de combustible. La longitud del trayecto que va a realizar (“distancia hasta el destino”) se puede incluir en el código para simplificar las cosas; puede utilizar la distancia media de su viaje. Realizará algunos cálculos basados en la entrada (“¿A cuántos litros equivale ese porcentaje de lleno? ¿Es la distancia media del viaje mayor que los kilómetros que puede recorrer, dados los litros que tiene y su promedio de ‘kilómetros por litro’?”). Usted realiza estos cálculos y envía los mensajes en eventos.

Hasta ahora tiene dos estados y una entrada. Necesita un evento en el primer estado que realice los cálculos en función de la entrada y decida si se pasa al segundo estado. Eso es un evento de transición. (Los `transitionEvents` están en una lista de eventos `onInput` del estado. Al recibirse una entrada en este primer estado, el evento realiza una transición al segundo estado, si se cumple

la `condition` del evento). Al llegar al segundo estado, envía el mensaje en cuanto entra en él. (Usted utiliza un evento `onEnter`. Al entrar en el segundo estado, este evento envía el mensaje. No es necesario esperar a que llegue otra entrada). Hay otros tipos de eventos, pero esto es todo lo que necesita para un ejemplo sencillo.

Los otros tipos de eventos son `onExit` y `onInput`. En cuanto se recibe una entrada, y si se cumple la condición, un evento `onInput` realiza las acciones especificadas. Cuando una operación sale de su estado actual y se cumple la condición, el evento `onExit` realiza las acciones especificadas.

¿Le falta algo? Sí, ¿cómo se vuelve al primer estado “el coche no necesita combustible”? Después de llenar el depósito de gasolina, la entrada muestra el depósito lleno. En el segundo estado necesita un evento de transición para volver al primer estado, que ocurra cuando se reciba la entrada (en los eventos `onInput`: del segundo estado). Debería volver al primer estado si los cálculos muestran que ahora tiene suficiente gasolina para llegar a donde quiere ir.

Eso es lo básico. Algunos modelos de detectores se vuelven más complejos añadiendo estados que reflejan entradas importantes, no solo posibles acciones. Por ejemplo, podría tener tres estados en un modelo de detector que vigile la temperatura: un estado “normal”, un estado “demasiado caliente” y un estado “problema potencial”. Usted pasa al estado de problema potencial cuando la temperatura supera un determinado nivel, pero aún no es demasiado caliente. No querrá enviar una alarma a menos que permanezca a esta temperatura durante más de 15 minutos. Si la temperatura vuelve a la normalidad antes de ese periodo, el detector vuelve al estado normal. Si el temporizador expira, el detector pasa al estado demasiado caliente y envía una alarma, solo por precaución. Podría hacer lo mismo utilizando variables y un conjunto más complejo de condiciones de evento. Sin embargo, a menudo es más fácil utilizar otro estado para, en efecto, almacenar los resultados de sus cálculos.

¿Cómo puede saber si necesita una instancia, o varias, de un detector?

Para decidir cuántas instancias necesita, hágase la siguiente pregunta: “¿Qué le interesa saber?”. Digamos que quiere saber qué tiempo hace hoy. ¿Está lloviendo (estado)? ¿Necesita llevar un paraguas (acción)? Puede tener un sensor que informe de la temperatura, otro que informe de la humedad y otros que informen de la presión barométrica, la velocidad y dirección del viento y la precipitación. Sin embargo, debe monitorear todos estos sensores a la vez para determinar el estado del tiempo (lluvia, nieve, nublado, soleado) y la acción apropiada por realizar (llevar un paraguas o aplicarse protector solar). Sin importar el número de sensores, lo que quiere es una única instancia de detector para monitorear el estado del tiempo e informarle de las medidas que debe tomar.

Pero si usted es el meteorólogo de su región, podría tener varias instancias de estos conjuntos de sensores, situados en diferentes lugares de la región. La gente de cada lugar necesita saber qué

tiempo hace en ese lugar. En este caso, necesita múltiples instancias de su detector. Los datos comunicados por cada sensor en cada ubicación deben incluir un campo que usted haya designado como campo key. Este campo permite a AWS IoT Events crear una instancia del detector para la zona y, a continuación, seguir dirigiendo esta información a esa instancia de detector a medida que siga llegando. ¡Se acabaron los cabellos estropeados o las narices quemadas por el sol!

Básicamente, necesita una única instancia de detector si tiene una sola situación (un proceso o una ubicación) que monitorear. Si tiene muchas situaciones (ubicaciones, procesos) que monitorear, necesitará varias instancias del detector.

Cree un AWS IoT Events detector para dos estados mediante CLI

En este ejemplo, utilizamos los AWS CLI comandos de AWS IoT Events APIs uso para crear un detector que modele dos estados de un motor: un estado normal y un estado de sobrepresión.

Cuando la presión medida en el motor supera un determinado umbral, el modelo pasa al estado de sobrepresión y envía un mensaje de Amazon Simple Notification Service (Amazon SNS) para avisar al técnico de la situación. Cuando la presión cae por debajo del umbral durante tres lecturas de presión consecutivas, el modelo vuelve al estado normal y envía otro mensaje Amazon SNS como confirmación de que la condición ha desaparecido. Necesitamos tres lecturas consecutivas por debajo del umbral de presión para eliminar posibles tartamudeos de mensajes de sobrepresión/normalidad en caso de una fase de recuperación no lineal o una lectura de recuperación anómala puntual.

A continuación, se resumen los pasos para crear el detector.

Creación de entradas.

Para supervisar sus dispositivos y procesos, deben tener una forma de transferir datos de telemetría a AWS IoT Events. Esto se hace enviando mensajes como entradas a AWS IoT Events. Puede hacer esto de varias formas:

- Usa la [BatchPutMessage](#) operación. Este método es fácil, pero requiere que sus dispositivos o procesos puedan acceder a la AWS IoT Events API a través de un SDK o el AWS CLI.
- En AWS IoT Core, escribe una regla de [AWS IoT Events acción](#) para el motor de AWS IoT Core reglas al que reenvía los datos de tus mensajes. AWS IoT Events Esto identifica la entrada por su nombre. Utilice este método si sus dispositivos o procesos pueden enviar mensajes a través AWS IoT Core de ellos o ya lo están haciendo. Este método suele requerir menos potencia de cálculo de un dispositivo.

- En AWS IoT Analytics, utilice la [CreateDataset](#) operación para crear un conjunto de datos `contentDeliveryRules` que especifique la AWS IoT Events entrada, donde el contenido del conjunto de datos se envía automáticamente. Utilice este método si desea controlar sus dispositivos o procesos en función de datos agregados o analizados en AWS IoT Analytics.

Para que sus dispositivos puedan enviar datos de este modo, debe definir una o más entradas. Para ello, asigne un nombre a cada entrada y especifique qué campos de los datos del mensaje entrante monitorea la entrada.

Crear un modelo de detector

Cree un modelo de detector (un modelo de su equipo o proceso) utilizando estados. Para cada estado, defina una lógica condicional (booleana) que evalúe las entradas para detectar eventos significativos. Cuando se detecta un evento, puede cambiar el estado o iniciar acciones personalizadas o predefinidas mediante otros AWS servicios. Puede definir eventos adicionales que inicien acciones al entrar o salir de un estado y, opcionalmente, al cumplirse una condición.

Supervisión de varios dispositivos o procesos

Si monitorea varios dispositivos o procesos y desea realizar un seguimiento de cada uno de ellos por separado, especifique un campo en cada entrada que identifique el dispositivo o proceso en particular del que proviene la entrada. Consulte el campo `key` en `CreateDetectorModel`. Al identificarse un nuevo dispositivo (se ve un nuevo valor en el campo de entrada identificado por la `key`), se crea una instancia de detector. La nueva instancia de detector sigue respondiendo a las entradas procedentes de ese dispositivo en particular hasta que se actualice o elimine su modelo de detector. Tendrá tantos detectores únicos (instancias) como valores únicos haya en los campos `key` de entrada.

Monitoreo de un único dispositivo o proceso

Si supervisa un único proceso (incluso con varios dispositivos o subprocesos enviando entradas), usted no especifica un campo `key` de identificación único. En este caso, se crea un único detector (instancia) al llegar la primera entrada. Por ejemplo, podría tener sensores de temperatura en cada habitación de una casa, pero solo una unidad de HVAC para climatizar toda la casa. Por tanto, solo puede controlarla como un proceso único, incluso si el ocupante de cada habitación quiere que prevalezca su voto (opinión).

Envío de mensajes desde sus dispositivos o procesos como entradas a su modelo de detector

Describimos las diversas formas de enviar un mensaje desde un dispositivo o proceso como entrada a un AWS IoT Events detector en entradas. Después de crear las entradas y construir el modelo de detector, ya está listo para empezar a enviar datos.

Note

Al crear un modelo de detector, o actualizar uno existente, transcurren varios minutos antes de que el modelo de detector nuevo o actualizado comience a recibir mensajes y a crear detectores (instancias). Al actualizar el modelo de detector, es posible que durante este periodo siga observando un comportamiento basado en la versión anterior.

Temas

- [Cree una AWS IoT Events entrada para capturar los datos del dispositivo](#)
- [Cree un modelo de detector para representar los estados de los dispositivos en AWS IoT Events](#)
- [Envíe mensajes como entradas a un detector en AWS IoT Events](#)

Cree una AWS IoT Events entrada para capturar los datos del dispositivo

Al configurar las entradas AWS IoT Events, puede aprovecharlas AWS CLI para definir la forma en que sus dispositivos comunican los datos de los sensores. Por ejemplo, si tus dispositivos envían mensajes en formato JSON con identificadores del motor y lecturas de los sensores, puedes capturar estos datos creando una entrada que mapee atributos específicos de los mensajes, como la presión y el identificador del motor. El proceso comienza definiendo una entrada en un archivo JSON, especificando los puntos de datos relevantes y utilizándolos AWS CLI para registrar la entrada. AWS IoT Events Esto permite monitorear y responder AWS IoT a condiciones críticas en función de los datos de los sensores en tiempo real.

Como ejemplo, suponga que sus dispositivos envían mensajes con el siguiente formato.

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

Puede crear una entrada para capturar los pressure datos y los motorid (que identifique el dispositivo específico que envió el mensaje) mediante el siguiente AWS CLI comando.

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

El archivo `pressureInput.json` contiene lo siguiente.

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

Al crear sus propias entradas, recuerde recopilar primero los mensajes de ejemplo como archivos JSON de sus dispositivos o procesos. Puede utilizarlos para crear una entrada desde la consola o la CLI.

Cree un modelo de detector para representar los estados de los dispositivos en AWS IoT Events

En [Cree una AWS IoT Events entrada para capturar los datos del dispositivo](#) creó una `input` basada en un mensaje que informa de los datos de presión de un motor. Para continuar con el ejemplo, aquí tiene un modelo de detector que responde a un evento de sobrepresión en un motor.

Usted crea dos estados: “Normal” y “Dangerous”. Cada detector (instancia) entra en el estado “Normal” cuando se crea. La instancia se crea al llegar una entrada con un valor único para la key “motorid”.

Si la instancia del detector recibe una lectura de presión de 70 o superior, entra en el estado “Dangerous” y envía un mensaje Amazon SNS a modo de advertencia. Si las lecturas de presión vuelven a la normalidad (inferior a 70) durante tres entradas consecutivas, el detector vuelve al estado “Normal” y envía otro mensaje Amazon SNS indicando todo normal.

En este ejemplo de modelo de detector se supone que ha creado dos temas de Amazon SNS cuyos nombres de recursos de Amazon (ARNs) se muestran en la definición como “targetArn”: “arn:aws:sns:us-east-1:123456789012:underPressureAction” y “targetArn”: “arn:aws:sns:us-east-1:123456789012:pressureClearedAction”

Para obtener más información, consulte la [Guía para desarrolladores de Amazon Simple Notification Service](#) y, más específicamente, la documentación de la [CreateTopic](#) operación en la referencia de la API de Amazon Simple Notification Service.

En este ejemplo también se supone que ha creado un rol AWS Identity and Access Management (de IAM) con los permisos adecuados. El ARN de este rol se muestra en la definición del modelo de detector como "roleArn": "arn:aws:iam::123456789012:role/IoTEventsRole". Siga los pasos que se indican en [Configuración de permisos para AWS IoT Events](#) para crear este rol y copie el ARN del rol en el lugar apropiado de la definición del modelo de detector.

Puede crear el modelo de detector mediante el siguiente AWS CLI comando.

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

El archivo "motorDetectorModel.json" contiene lo siguiente.

```
{
  "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"  
}
```

Envíe mensajes como entradas a un detector en AWS IoT Events

Ahora ha definido una entrada que identifica los campos importantes en los mensajes enviados desde un dispositivo (consulte [Cree una AWS IoT Events entrada para capturar los datos del dispositivo](#)). En la sección anterior, creó un detector `model` que responde a un evento de sobrepresión en un motor (consulte [Cree un modelo de detector para representar los estados de los dispositivos en AWS IoT Events](#)).

Para completar el ejemplo, envíe mensajes desde un dispositivo (en este caso un ordenador con la AWS CLI instalada) como entradas al detector.

Note

Al crear un modelo de detector o actualizar uno existente, transcurren varios minutos antes de que el modelo de detector nuevo o actualizado comience a recibir mensajes y a crear detectores (instancias). Si usted actualiza el modelo de detector, es posible que durante este periodo siga observando un comportamiento basado en la versión anterior.

Usa el siguiente AWS CLI comando para enviar un mensaje con datos que superen el umbral.

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

El archivo `highPressureMessage.json` contiene lo siguiente.

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

```
    }  
  ]  
}
```

Debe cambiar el `messageId` en cada mensaje enviado. Si no lo cambias, el AWS IoT Events sistema deduplicará los mensajes. AWS IoT Events ignora un mensaje si es `messageID` igual a otro mensaje que se envió en los últimos cinco minutos.

A este punto, se crea un detector (instancia) para monitorear los eventos del motor "Fulton-A32". Este detector entra en el estado "Normal" cuando se crea. Pero como hemos enviado un valor de presión superior al umbral, pasa de inmediato al estado "Dangerous". Al hacerlo, el detector envía un mensaje al punto de conexión de Amazon SNS con ARN `arn:aws:sns:us-east-1:123456789012:underPressureAction`.

Ejecute el siguiente AWS CLI comando para enviar un mensaje con datos que estén por debajo del umbral de presión.

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

El archivo `normalPressureMessage.json` contiene lo siguiente.

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

Debe cambiar el `messageId` en el archivo cada vez que invoque el comando `BatchPutMessage` en un periodo de cinco minutos. Envíe el mensaje dos veces más. Después de enviar tres veces el mensaje, el detector (instancia) del motor "Fulton-A32" envía un mensaje al punto de conexión `arn:aws:sns:us-east-1:123456789012:pressureClearedAction` de Amazon SNS y vuelve a entrar en el estado "Normal".

 Note

Puede enviar varios mensajes a la vez con `BatchPutMessage`. Sin embargo, no se garantiza el orden en que se procesan estos mensajes. Para garantizar que los mensajes (entradas) se procesen en orden, envíelos de uno en uno y espere una respuesta satisfactoria cada vez que se llame a la API.

A continuación, se muestran ejemplos de cargas de mensajes SNS creadas por el ejemplo de modelo de detector descrito en esta sección.

en caso de “Superación del umbral de presión”

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

en caso de “Restablecimiento de la presión normal”

```
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"
}
```

Si ha definido algún temporizador, su estado actual también se muestra en las cargas de los mensajes SNS.

Las cargas de los mensajes contienen información sobre el estado del detector (instancia) en el momento en que se envió el mensaje (es decir, en el momento en que se ejecutó la acción SNS). Puede utilizar la operación https://docs.aws.amazon.com/iotevents/latest/apireference/API_iotevents-data_DescribeDetector.html para obtener información similar sobre el estado del detector.

AWS IoT Events restricciones y limitaciones del modelo de detector

Es importante tener en cuenta los siguientes aspectos al crear un modelo de detector.

Cómo usar el campo **actions**

El campo **actions** es una lista de objetos. Puede tener más de un objeto, pero solo se permite una acción en cada objeto.

Example

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

Cómo usar el campo **condition**

La **condition** es obligatoria para **transitionEvents** y opcional en otros casos.

Si el campo **condition** no está presente, equivale a **"condition": true**.

El resultado de la evaluación de una expresión de condición debe ser un valor booleano. Si el resultado no es un valor booleano, equivale a **false** y no se iniciarán las **actions** ni la transición al **nextState** en el evento.

Disponibilidad de valores variables

Por defecto, si se fija el valor de una variable en un evento, no se dispone de su nuevo valor ni se utiliza para evaluar condiciones en otros eventos del mismo grupo. El nuevo valor no está disponible ni se utiliza en una condición de evento en el mismo campo **onInput**, **onEnter** o **onExit**.

Establezca el parámetro **evaluationMethod** en la definición del modelo de detector para cambiar este comportamiento. Si se establece **evaluationMethod** en **SERIAL**, las variables se actualizan y las condiciones de evento se evalúan en el orden en que estén definidos los eventos. Caso contrario, si se establece **evaluationMethod** en **BATCH** o lo asume por defecto, las variables dentro de un estado se actualizan y los eventos dentro de un estado se realizan solo después de que se hayan evaluado todas las condiciones del evento.

En el estado "Dangerous", en el campo onInput, "\$variable.pressureThresholdBreach" se decrementa en uno en el evento "Pressure Okay" al cumplirse la condición (cuando la entrada actual tiene una presión menor o igual que 70).

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

El detector debería volver al estado "Normal" cuando "\$variable.pressureThresholdBreach" llegue a 0 (es decir, cuando el detector haya recibido tres lecturas de presión consecutivas menores o iguales que 70). El evento "BackToNormal" en transitionEvents debe comprobar que "\$variable.pressureThresholdBreach" sea menor o igual que 1 (no 0) y además volver a verificar que el valor actual dado por "\$input.PressureInput.sensorData.pressure" sea menor o igual que 70.

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

Caso contrario, si la condición comprueba solo el valor de la variable, dos lecturas normales seguidas de una lectura de sobrepresión cumplirían la condición y se volvería al estado "Normal". La condición mira el valor que se le dio a

"\$variable.pressureThresholdBreached" durante la última vez que se procesó una entrada. El valor de la variable se restablece a 3 en el evento "Overpressurized", pero recuerde que este nuevo valor aún no está disponible para ninguna condition.

Por defecto, cada vez que un control entra en el campo onInput, una condition solo puede ver el valor de una variable tal y como era al inicio del procesamiento de la entrada, antes de que cualquier acción especificada en onInput lo modifique. Lo mismo ocurre para onEnter y onExit. Cualquier cambio realizado en una variable al entrar o salir del estado no está disponible para otras condiciones especificadas en los mismos campos onEnter o onExit.

Latencia al actualizar un modelo de detector

Si actualiza, elimina y vuelve a crear un modelo de detector (consulte [UpdateDetectorModel](#)), pasará un tiempo antes de que se eliminen todos los detectores generados (instancias) y se utilice el nuevo modelo para volver a crear los detectores. Se recrean una vez que el nuevo modelo de detector entra en función y llegan nuevas entradas. Durante este tiempo, es posible que los detectores generados por la versión anterior del modelo de detector sigan procesando entradas. Durante este período, es posible que siga recibiendo las alertas definidas por el modelo de detector anterior.

Espacios en las claves de entrada

Se permiten espacios en las claves de entrada, pero las referencias a la clave deben estar entre tildes graves, tanto en la definición del atributo de entrada como cuando se hace referencia al valor de la clave en una expresión. Por ejemplo, dada una carga de mensaje como la siguiente:

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

Utilice lo siguiente para definir la entrada.

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

```
{ "jsonPath": "`motor id`" }  
]  
}  
}
```

En una expresión condicional, debe hacer referencia al valor de cualquiera de estas claves utilizando también tildes graves.

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

Un ejemplo comentado: control de temperatura de HVAC con AWS IoT Events

Algunos de los siguientes archivos JSON de ejemplo tienen comentarios en línea, lo que los convierte en JSON no válidos. Las versiones completas de estos ejemplos, sin comentarios, están disponibles en [Ejemplo: usar el control de temperatura de HVAC con AWS IoT Events](#).

En este ejemplo se implementa un modelo de control de termostato que le permite hacer lo siguiente.

- Definir un único modelo de detector que pueda utilizarse para supervisar y controlar múltiples áreas. Se crea una instancia de detector para cada área.
- Ingerir datos de temperatura de múltiples sensores en cada área de control.
- Cambiar el punto de consigna de temperatura para un área.
- Establecer parámetros operativos para cada área y restablecerlos mientras se utiliza la instancia.
- Añadir o eliminar de forma dinámica sensores en un área.
- Especificar un tiempo de ejecución mínimo para proteger las unidades de calefacción y refrigeración.
- Rechazar lecturas anómalas de los sensores.
- Definir puntos de consigna de emergencia que activen de inmediato la calefacción o la refrigeración si alguno de los sensores informa de una temperatura superior o inferior a un umbral determinado.
- Notificar lecturas anómalas y picos de temperatura.

Temas

- [Introduzca las definiciones para los modelos de detectores en AWS IoT Events](#)

- [Cree una definición AWS IoT Events de modelo de detector](#)
- [Se utiliza BatchUpdateDetector para actualizar un modelo AWS IoT Events de detector](#)
- [Se utiliza BatchPutMessage para entradas en AWS IoT Events](#)
- [Ingiera los mensajes MQTT en AWS IoT Events](#)
- [Genere mensajes de Amazon SNS en AWS IoT Events](#)
- [Configure la API en DescribeDetector AWS IoT Events](#)
- [Utilice el motor de AWS IoT Core reglas para AWS IoT Events](#)

Introduzca las definiciones para los modelos de detectores en AWS IoT Events

Queremos crear un modelo de detector que podamos utilizar para monitorear y controlar la temperatura en varias áreas diferentes. Cada área puede tener varios sensores que informen de la temperatura. Suponemos que cada área cuenta con una unidad de calefacción y otra de refrigeración que se pueden encender o apagar para controlar la temperatura del área. Cada área está controlada por una instancia del detector.

Dado que las distintas áreas que monitoreamos y controlamos podrían tener características diferentes que exijan parámetros de control distintos, definimos las 'seedTemperatureInput' para que proporcionen esos parámetros para cada área. Cuando enviamos uno de estos mensajes de entrada a AWS IoT Events, se crea una nueva instancia del modelo de detector que tiene los parámetros que queremos utilizar en esa área. Esta es la definición de esta entrada.

Comando de la CLI:

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

Archivo: 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" }
]
}
}
```

Respuesta:

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

Notas

- Se crea una nueva instancia del detector para cada 'areaId' único recibido en cualquier mensaje. Consulte el campo 'key' en la definición de 'areaDetectorModel'.
- La temperatura media puede variar de la 'desiredTemperature' en un 'allowedError' antes de que se activen las unidades de calefacción o refrigeración del área.
- Si cualquier sensor notifica una temperatura superior a 'rangeHigh', el detector informa de un pico y de inmediato pone en marcha la unidad de refrigeración.
- Si cualquier sensor notifica una temperatura inferior a 'rangeLow', el detector informa de un pico y de inmediato pone en marcha la unidad de calefacción.
- Si cualquier sensor notifica una temperatura superior a 'anomalousHigh' o inferior a 'anomalousLow', el detector informa de una lectura anómala de sensor, pero ignora la lectura de temperatura informada.
- El 'sensorCount' le indica al detector cuántos sensores están emitiendo información del área. El detector calcula la temperatura media del área dando el factor de ponderación apropiado a

cada lectura de temperatura que recibe. Por ello, el detector no tendrá que hacer un seguimiento de lo que informa cada sensor y el número de sensores se puede cambiar de manera dinámica según sea necesario. Sin embargo, si un sensor individual deja de emitir, el detector no lo sabrá o no tendrá en cuenta esta circunstancia. Le recomendamos que cree otro modelo de detector específico para monitorear el estado de conexión de cada sensor. Disponer de dos modelos de detector complementarios simplifica el diseño de ambos.

- El valor de 'noDelay' puede ser true o false. Una unidad de calefacción o refrigeración, tras encenderse, debe permanecer encendida durante un tiempo mínimo determinado para proteger la integridad de la unidad y alargar su vida útil. Si 'noDelay' se configura en false, la instancia del detector aplica un retardo antes de apagar las unidades de refrigeración y calefacción a fin de garantizar que funcionen durante el tiempo mínimo. El número de segundos de retardo se ha codificado en la definición del modelo de detector porque no podemos utilizar un valor variable para establecer un temporizador.

Se utiliza 'temperatureInput' para transmitir los datos del sensor a una instancia del detector.

Comando de la CLI:

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

Archivo: temperatureInput.json

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

Respuesta:

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

Notas

- Una instancia del detector de ejemplo no utiliza el 'sensorId' para controlar ni monitorear un sensor de manera directa. Se transfiere automáticamente a las notificaciones enviadas por la instancia del detector. A partir de ahí, se lo puede utilizarse para identificar sensores que estén fallando (p. ej., un sensor que envía regularmente lecturas anómalas podría estar a punto de fallar) o que han dejado de emitir (cuando se utiliza como entrada de un modelo de detector adicional que controla el latido del dispositivo). El 'sensorId' también puede ayudar a identificar zonas cálidas o frías en un área si sus lecturas difieren regularmente de la media.
- El 'areaId' se utiliza para dirigir los datos del sensor a la instancia de detector apropiada. Se crea una instancia de detector para cada 'areaId' único recibido en cualquier mensaje. Consulte el campo 'key' en la definición de 'areaDetectorModel'.

Cree una definición AWS IoT Events de modelo de detector

El ejemplo de 'areaDetectorModel' tiene comentarios en línea.

Comando de la CLI:

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

Archivo: 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
lost.
                            // 'timeout()' is used in such mixed conditionals, its value is

                            "variableName": "goodToGo",
                            "value": "false"
                        }
                    }
                ]
            }
        ]
    }
}

```

```

    ]
  },
  {
    "eventName": "dontDelay",
    "condition": "$variable.noDelay == true",
    // If the heating/cooling unit shutoff delay is not used, no need to
wait.
    "actions": [
      {
        "setVariable": {
          "variableName": "goodToGo",
          "value": "true"
        }
      }
    ]
  },
  {
    "eventName": "beenHere",
    "condition": "true",
    "actions": [
      {
        "setVariable": {
          "variableName": "enteringNewState",
          "value": "false"
        }
      }
    ]
  }
],
},
"onInput": {
  "events": [
    // These are events that occur when an input is received (if the condition
is
    // satisfied), but don't cause a transition to another state.
    {
      "eventName": "whatWasInput",
      "condition": "true",
      "actions": [
        {
          "setVariable": {
            "variableName": "sensorId",
            "value": "$input.temperatureInput.sensorId"
          }
        }
      ]
    }
  ]
}

```

```

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

```

```

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

```

```

    ],
    "nextState": "cooling"
  },

  {
    "eventName": "lowTemperatureSpike",
    "condition": "$input.temperatureInput.sensorData.temperature <
$variable.rangeLow",
    "actions": [
      {
        "iotTopicPublish": {
          "mqttTopic": "temperatureSensor/spike"
        }
      },
      {
        "sns": {
          "targetArn": "arn:aws:sns:us-west-2:123456789012:cool0ff"
        }
      },
      {
        "sns": {
          "targetArn": "arn:aws:sns:us-west-2:123456789012:heat0n"
        }
      },
      {
        "iotTopicPublish": {
          "mqttTopic": "hvac/Cooling/Off"
        }
      },
      {
        "iotTopicPublish": {
          "mqttTopic": "hvac/Heating/On"
        }
      },
      {
        "setVariable": {
          "variableName": "enteringNewState",
          "value": "true"
        }
      }
    ],
    "nextState": "heating"
  },

```

```

    {
      "eventName": "desiredTemperature",
      "condition": "((((($variable.averageTemperature * ($variable.sensorCount
- 1)) + $input.temperatureInput.sensorData.temperature) / $variable.sensorCount) <=
($variable.desiredTemperature - $variable.allowedError)) && $variable.goodToGo ==
true",
      "actions": [
        {
          "sns": {
            "targetArn": "arn:aws:sns:us-west-2:123456789012:cool0ff"
          }
        },
        {
          "iotTopicPublish": {
            "mqttTopic": "hvac/Cooling/Off"
          }
        }
      ],
      "nextState": "idle"
    }
  ]
},

```

```

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

```

```

        }
      ]
    },
    {
      "eventName": "dontDelay",
      "condition": "$variable.noDelay == true",
      "actions": [
        {
          "setVariable": {
            "variableName": "goodToGo",
            "value": "true"
          }
        }
      ]
    },
    {
      "eventName": "beenHere",
      "condition": "true",
      "actions": [
        {
          "setVariable": {
            "variableName": "enteringNewState",
            "value": "false"
          }
        }
      ]
    }
  ]
},
"onInput": {
  "events": [
    {
      "eventName": "whatWasInput",
      "condition": "true",
      "actions": [
        {
          "setVariable": {
            "variableName": "sensorId",
            "value": "$input.temperatureInput.sensorId"
          }
        }
      ],
      {
        "setVariable": {

```

```

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

```

```

    ]
  }
],
"transitionEvents": [
  {
    "eventName": "anomalousInputArrived",
    "condition": "$input.temperatureInput.sensorData.temperature >=
$variable.anomalousHigh || $input.temperatureInput.sensorData.temperature <=
$variable.anomalousLow",
    "actions": [
      {
        "iotTopicPublish": {
          "mqttTopic": "temperatureSensor/anomaly"
        }
      }
    ],
    "nextState": "heating"
  },
  {
    "eventName": "highTemperatureSpike",
    "condition": "$input.temperatureInput.sensorData.temperature >
$variable.rangeHigh",
    "actions": [
      {
        "iotTopicPublish": {
          "mqttTopic": "temperatureSensor/spike"
        }
      },
      {
        "sns": {
          "targetArn": "arn:aws:sns:us-west-2:123456789012:heatOff"
        }
      },
      {
        "sns": {
          "targetArn": "arn:aws:sns:us-west-2:123456789012:coolOn"
        }
      },
      {
        "iotTopicPublish": {
          "mqttTopic": "hvac/Heating/Off"
        }
      }
    ],
  },

```

```

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

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

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

```

```

        }
      }
    ],
    "nextState": "idle"
  }
]
}

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

```

Respuesta:

```

{
  "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"
  }
}

```

Se utiliza BatchUpdateDetector para actualizar un modelo AWS IoT Events de detector

Puede utilizar la operación BatchUpdateDetector para poner una instancia de detector en un estado conocido, incluyendo valores de temporizadores y de variables. En el siguiente ejemplo, la operación BatchUpdateDetector restablece los parámetros operativos de un área que está bajo monitoreo y control de temperatura. Esta operación le permite hacerlo sin tener que eliminar y recrear ni actualizar el modelo de detector.

Comando de la CLI:

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

Archivo: 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": [
  ]
}
]
}

```

Respuesta:

```

{
  "batchUpdateDetectorErrorEntries": []
}

```

Se utiliza BatchPutMessage para entradas en AWS IoT Events

Example 1

Utilice la operación BatchPutMessage para enviar un mensaje "seedTemperatureInput" que establezca los parámetros operativos de un área determinada bajo monitoreo y control de la temperatura. Cualquier mensaje AWS IoT Events que reciba un mensaje nuevo "areaId" provoca la creación de una nueva instancia de detector. Sin embargo, la nueva instancia de detector no cambiará de estado a "idle" ni comenzará a supervisar la temperatura ni a controlar las unidades de calefacción o refrigeración hasta que se reciba un mensaje "seedTemperatureInput" para la nueva área.

Comando de la CLI:

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

Archivo: 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}"
    }
  ]
}
```

Respuesta:

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

Example

2

Utilice la operación `BatchPutMessage` para enviar un mensaje "temperatureInput" a fin de notificar los datos de un sensor de temperatura en un área de control y monitoreo determinada.

Comando de la CLI:

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

Archivo: temperatureExample.json

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

Respuesta:

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

Example 3

Utilice la operación `BatchPutMessage` para enviar un mensaje "seedTemperatureInput" a fin de cambiar el valor de la temperatura deseada en un área determinada.

Comando de la CLI:

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

Archivo: seedSetDesiredTemp.json

```
{
```

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

Respuesta:

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

Ingiera los mensajes MQTT en AWS IoT Events

Si los recursos de computación de sus sensores no pueden usar la "BatchPutMessage" API, pero pueden enviar sus datos al agente de AWS IoT Core mensajes mediante un cliente MQTT ligero, puede crear una regla AWS IoT Core temática para redirigir los datos de los mensajes a una AWS IoT Events entrada. La siguiente es una definición de una regla de AWS IoT Events tema que toma los campos "areaId" y de "sensorId" entrada del tema MQTT y el "sensorData.temperature" campo del campo de carga del mensaje e "temp" incorpora estos datos a nuestro. AWS IoT Events "temperatureInput"

Comando de la CLI:

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

Archivo: seedSetDesiredTemp.json

```
{  
  "ruleName": "temperatureTopicRule",  
  "topicRulePayload": {  
    "sql": "SELECT topic(3) as areaId, topic(4) as sensorId, temp as  
sensorData.temperature FROM 'update/temperature/#'",  
    "description": "Ingest temperature sensor messages into IoT Events",  
    "actions": [  
      {  

```

```
    "iotEvents": {
      "inputName": "temperatureInput",
      "roleArn": "arn:aws:iam::123456789012:role/service-role/anotherRole"
    }
  ],
  "ruleDisabled": false,
  "awsIotSqlVersion": "2016-03-23"
}
```

Respuesta: [none]

Si el sensor envía un mensaje sobre el tema "update/temperature/Area51/03" con la siguiente carga.

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

Esto hace que los datos se ingieran AWS IoT Events como si se hubiera realizado la siguiente llamada a la "BatchPutMessage" API.

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

Archivo: spoofExample.json

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

Genere mensajes de Amazon SNS en AWS IoT Events

Los siguientes son ejemplos de mensajes SNS generados por la instancia del detector "Area51".

AWS IoT Events puede integrarse con Amazon SNS para generar y publicar notificaciones en función de los eventos detectados. En esta sección se muestra cómo una instancia de AWS IoT Events detector, específicamente el detector «Area51», genera mensajes de Amazon SNS. Estos ejemplos muestran la estructura y el contenido de las notificaciones de Amazon SNS activadas por varios estados y eventos dentro del AWS IoT Events detector, lo que ilustra el poder de la combinación con Amazon AWS IoT Events SNS para generar alertas y comunicaciones en tiempo real.

```
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"}
```

Configure la API en DescribeDetector AWS IoT Events

La `DescribeDetector` API de AWS IoT Events entrada le permite recuperar información detallada sobre una instancia de detector específica. Esta operación proporciona información sobre el estado actual, los valores de las variables y los temporizadores activos de un detector. Al usar esta API, puede monitorear el estado en tiempo real de sus AWS IoT Events detectores, lo que facilita la depuración, el análisis y la administración de sus flujos de trabajo de procesamiento de eventos de IoT.

Comando de la CLI:

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

Respuesta:

```
{
  "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"
}
}
```

Utilice el motor de AWS IoT Core reglas para AWS IoT Events

Las siguientes reglas vuelven a publicar los mensajes AWS IoT Core MQTT como mensajes de solicitud de actualización paralelos. Suponemos que AWS IoT Core las cosas están definidas para una unidad de calefacción y una unidad de refrigeración para cada área controlada por el modelo de detector. En este ejemplo, hemos definido cosas denominadas "Area51HeatingUnit" y "Area51CoolingUnit".

Comando de la CLI:

```
aws iot create-topic-rule --cli-input-json file://ADMSHadowCool0ffRule.json
```

Archivo: ADMSHadowCool0ffRule.json

```
{
  "ruleName": "ADMSHadowCool0ff",
  "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"
        }
      }
    ]
  }
}
```

Respuesta: [vacío]

Comando de la CLI:

```
aws iot create-topic-rule --cli-input-json file://ADMSHadowCool0nRule.json
```

Archivo: 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": [
      {
        "republsh": {
          "topic": "$$aws/things/${payload.detector.keyValue}CoolingUnit/shadow/update",
          "roleArn": "arn:aws:iam::123456789012:role/service-role/ADMShadowRole"
        }
      }
    ]
  }
}
```

Respuesta: [vacío]

Comando de la CLI:

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

Archivo: 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": [
      {
        "republsh": {
```

```

        "topic": "$$aws/things/${payload.detector.keyValue}HeatingUnit/shadow/
update",
        "roleArn": "arn:aws:iam::123456789012:role/service-role/ADMSHadowRole"
    }
}
]
}
}

```

Respuesta: [vacío]

Comando de la CLI:

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

Archivo: 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"
        }
      }
    ]
  }
}

```

Respuesta: [vacío]

Acciones compatibles para recibir datos y activar acciones en AWS IoT Events

AWS IoT Events puede activar acciones cuando detecta un evento específico o un evento de transición. Puede definir acciones integradas para usar un temporizador o establecer una variable, o enviar datos a otros recursos de AWS . Aprenda a configurar y personalizar estas acciones para crear respuestas automatizadas a sus diversos eventos de IoT.

Note

Al definir una acción en un modelo de detector, puede utilizar expresiones para parámetros que sean de tipo cadena de datos. Para obtener más información, consulte [Expresiones](#).

AWS IoT Events admite las siguientes acciones que le permiten usar un temporizador o configurar una variable:

- [setTimer](#) para crear un temporizador.
- [resetTimer](#) para restablecer el temporizador.
- [clearTimer](#) para eliminar el temporizador.
- [setVariable](#) para crear una variable.

AWS IoT Events admite las siguientes acciones que le permiten trabajar con AWS los servicios:

- [iotTopicPublish](#) para publicar un mensaje en un tema de MQTT.
- [iotEvents](#) para enviar datos a AWS IoT Events como valor de entrada.
- [iotSiteWise](#) para enviar datos a una propiedad de recurso en AWS IoT SiteWise.
- [dynamoDB](#) para enviar datos a una tabla de Amazon DynamoDB.
- [dynamoDBv2](#) para enviar datos a una tabla de Amazon DynamoDB.
- [firehose](#) para enviar datos a una transmisión de Amazon Data Firehose.
- [lambda](#) para invocar una función de AWS Lambda .
- [sns](#) para enviar datos como notificación de inserción.
- [sqs](#) para enviar datos a una cola de Amazon SQS.

Utilice el temporizador AWS IoT Events incorporado y las acciones variables

AWS IoT Events admite las siguientes acciones que permiten utilizar un temporizador o configurar una variable:

- [setTimer](#) para crear un temporizador.
- [resetTimer](#) para restablecer el temporizador.
- [clearTimer](#) para eliminar el temporizador.
- [setVariable](#) para crear una variable.

Establecer la acción del temporizador

Set timer action

La acción `setTimer` le permite crear un temporizador con una duración en segundos.

More information (2)

Al crear un temporizador, debe especificar los siguientes parámetros.

timerName

El nombre del temporizador.

durationExpression

(Opcional) Duración, en segundos, del temporizador.

El resultado evaluado de una expresión de duración se redondea al número entero menor más cercano. Por ejemplo, si configura el temporizador en 60,99 segundos, el resultado evaluado de la expresión de duración es 60 segundos.

Para obtener más información, consulta [SetTimerAction](#) en la AWS IoT Events Referencia de la API de .

Restablecer la acción de temporizador

Reset timer action

La acción `resetTimer` le permite ajustar el temporizador en el resultado previamente evaluado de la expresión de duración.

More information (1)

Al restablecer un temporizador, debe especificar el siguiente parámetro.

timerName

El nombre del temporizador.

AWS IoT Events no reevalúa la expresión de duración al restablecer el temporizador.

Para obtener más información, consulta [ResetTimerAction](#) en la AWS IoT Events Referencia de la API de .

Eliminar la acción del temporizador

Clear timer action

La acción `clearTimer` le permite eliminar un temporizador existente.

More information (1)

Al eliminar un temporizador, debe especificar el siguiente parámetro.

timerName

El nombre del temporizador.

Para obtener más información, consulta [ClearTimerAction](#) en la AWS IoT Events Referencia de la API de .

Establecer una acción de variable

Set variable action

La acción `setVariable` le permite crear una variable con un valor específico.

More information (2)

Al crear una variable, debe especificar los siguientes parámetros.

variableName

El nombre de la variable.

value

El nuevo valor de la variable.

Para obtener más información, consulta [SetVariableAction](#) en la AWS IoT Events Referencia de la API de .

AWS IoT Events trabajando con otros servicios AWS

AWS IoT Events admite las siguientes acciones que le permiten trabajar con AWS los servicios:

- [iotTopicPublish](#) para publicar un mensaje en un tema de MQTT.
- [iotEvents](#) para enviar datos a AWS IoT Events como valor de entrada.
- [iotSiteWise](#) para enviar datos a una propiedad de recurso en AWS IoT SiteWise.
- [dynamoDB](#) para enviar datos a una tabla de Amazon DynamoDB.
- [dynamoDBv2](#) para enviar datos a una tabla de Amazon DynamoDB.
- [firehose](#) para enviar datos a una transmisión de Amazon Data Firehose.
- [lambda](#) para invocar una función de AWS Lambda .
- [sns](#) para enviar datos como notificación de inserción.
- [sqs](#) para enviar datos a una cola de Amazon SQS.

Important

- Debe elegir la misma AWS región para ambas AWS IoT Events y los AWS servicios con los que desea trabajar. Para ver una lista completa de las regiones compatibles, consulte [AWS IoT Events endpoints and quotas](#) en la Referencia general de Amazon Web Services.
- Debe usar la misma AWS región al crear otros AWS recursos para las AWS IoT Events acciones. Si cambias de AWS región, es posible que tengas problemas para acceder a los AWS recursos.

De forma predeterminada, AWS IoT Events genera una carga útil estándar en JSON para cualquier acción. Esta carga de acción contiene todos los pares atributo-valor que tienen la información sobre la instancia del modelo de detector y el evento que desencadenó la acción. Para configurar la carga de acción, puede usar una expresión de contenido. Para obtener más información, consulte [Expresiones para filtrar, transformar y procesar datos de eventos](#) y el tipo de datos de [Carga](#) en la Referencia de la API de AWS IoT Events .

AWS IoT Core

IoT topic publish action

La AWS IoT Core acción permite publicar un mensaje MQTT a través del intermediario de AWS IoT mensajes. Para ver una lista completa de las regiones compatibles, consulte [AWS IoT Core endpoints and quotas](#) en la Referencia general de Amazon Web Services.

El agente de AWS IoT mensajes conecta a AWS IoT los clientes mediante el envío de mensajes desde los clientes publicadores a los clientes suscritos. Para obtener más información, consulte [Protocolos de comunicación de dispositivos](#) en la Guía para AWS IoT desarrolladores.

More information (2)

Al publicar un mensaje MQTT, debe especificar los siguientes parámetros.

mqttTopic

El tema de MQTT que recibe el mensaje.

Puede definir el nombre de un tema de MQTT de forma dinámica en tiempo de ejecución mediante variables o valores de entrada creados en el modelo de detector.

payload

(Opcional) La carga predeterminada contiene todos los pares atributo-valor que tienen la información sobre la instancia del modelo detector y el evento que desencadenó la acción. También puede personalizar la carga. Para obtener más información consulte [Carga](#) en la Referencia de la API de AWS IoT Events .

Note

Asegúrese de que la política adjunta a su función AWS IoT Events de servicio conceda el `iot:Publish` permiso. Para obtener más información, consulte [Gestión de identidad y acceso para AWS IoT Events](#).

Para obtener más información, consulta [lotTopicPublishAction](#) en la AWS IoT Events Referencia de la API de .

AWS IoT Events

IoT Events action

La AWS IoT Events acción te permite enviar datos AWS IoT Events como entrada. Para ver una lista completa de las regiones compatibles, consulte [AWS IoT Events endpoints and quotas](#) en la Referencia general de Amazon Web Services.

AWS IoT Events le permite monitorizar sus equipos o flotas de dispositivos para detectar fallos o cambios en el funcionamiento y activar acciones cuando se produzcan dichos eventos. Para obtener más información, consulte [¿Qué es? AWS IoT Events](#) en la Guía para AWS IoT Events desarrolladores.

More information (2)

Al enviar datos a AWS IoT Events, debe especificar los siguientes parámetros.

inputName

El nombre de la AWS IoT Events entrada que recibe los datos.

payload

(Opcional) La carga predeterminada contiene todos los pares atributo-valor que tienen la información sobre la instancia del modelo detector y el evento que desencadenó la acción. También puede personalizar la carga. Para obtener más información consulte [Carga](#) en la Referencia de la API de AWS IoT Events .

Note

Asegúrese de que la política adjunta a su función AWS IoT Events de servicio conceda el `iotevents:BatchPutMessage` permiso. Para obtener más información, consulte [Gestión de identidad y acceso para AWS IoT Events](#).

Para obtener más información, consulta [IoTEventsAction](#) en la AWS IoT Events Referencia de la API de .

AWS IoT SiteWise

IoT SiteWise action

La AWS IoT SiteWise acción le permite enviar datos a una propiedad de un activo en AWS IoT SiteWise. Para ver una lista completa de las regiones compatibles, consulte [AWS IoT SiteWise endpoints and quotas](#) en la Referencia general de Amazon Web Services.

AWS IoT SiteWise es un servicio gestionado que le permite recopilar, organizar y analizar datos de equipos industriales a escala. Para obtener más información, consulte [¿Qué es AWS IoT SiteWise?](#) en la Guía del usuario de AWS IoT SiteWise .

More information (11)

Al enviar datos a una propiedad de un activo AWS IoT SiteWise, debe especificar los siguientes parámetros.

Important

Para recibir los datos, debe utilizar una propiedad de activo existente en AWS IoT SiteWise.

- Si utiliza la AWS IoT Events consola, debe especificar si desea `propertyAlias` identificar la propiedad del activo objetivo.
- Si utiliza la AWS CLI, debe especificar una `propertyAlias` o ambas `assetId` e `propertyId` identificar la propiedad del activo objetivo.

Para obtener más información, consulte [Mapping industrial data streams to asset properties](#) en la Guía del usuario de AWS IoT SiteWise .

propertyAlias

(Opcional) El alias de la propiedad de activo. También puede especificar una expresión.

assetId

(Opcional) El ID del activo que tiene la propiedad especificada. También puede especificar una expresión.

propertyId

(Opcional) ID de la propiedad de activo. También puede especificar una expresión.

entryId

(Opcional) Un identificador único para esta entrada. Puede utilizar el ID de entrada para realizar un seguimiento de qué entrada de datos provoca un error en caso de que se produzca un fallo. El valor predeterminado es un nuevo identificador único. También puede especificar una expresión.

propertyValue

Una estructura que contenga los detalles sobre el valor de propiedad.

quality

(Opcional) La calidad del valor de la propiedad de activo. El valor debe ser GOOD, BAD o UNCERTAIN. También puede especificar una expresión.

timestamp

(Opcional) Una estructura que contenga la información de marca temporal. Si no especifica este valor, el predeterminado es el momento del evento.

timeInSeconds

La marca temporal, en segundos, en formato de tiempo Unix. El rango válido es de 1 a 31556889864403199. También puede especificar una expresión.

offsetInNanos

(Opcional) El desplazamiento en nanosegundos convertido de `timeInSeconds`. El rango válido es de 0 a 999999999. También puede especificar una expresión.

value

Estructura que contiene un valor de propiedad de activo.

 Important

Debe especificar uno de los siguientes tipos de valor, dependiendo del valor `dataType` de la propiedad de activo especificada. Para obtener más información, consulta [AssetProperty](#) en la AWS IoT SiteWise Referencia de la API de .

booleanValue

(Opcional) El valor de la propiedad de activo es un valor booleano que debe ser `TRUE` o `FALSE`. También puede especificar una expresión. Si utiliza una expresión, el resultado evaluado debe ser un valor booleano.

doubleValue

(Opcional) El valor de la propiedad de activo es un valor doble. También puede especificar una expresión. Si utiliza una expresión, el resultado evaluado debe ser un valor doble.

integerValue

(Opcional) El valor de la propiedad de activo es un entero. También puede especificar una expresión. Si utiliza una expresión, el resultado evaluado debe ser un entero.

stringValue

(Opcional) El valor de la propiedad de activo es una cadena. También puede especificar una expresión. Si utiliza una expresión, el resultado evaluado debe ser una cadena.

 Note

Asegúrese de que la política asociada a su función de AWS IoT Events servicio le conceda el `iotsitewise:BatchPutAssetPropertyValue` permiso. Para obtener más información, consulte [Gestión de identidad y acceso para AWS IoT Events](#).

Para obtener más información, consulta [lotSiteWiseAction](#) en la AWS IoT Events Referencia de la API de .

Amazon DynamoDB

DynamoDB action

La acción Amazon DynamoDB le permite enviar datos a una tabla de DynamoDB. Una columna de la tabla de DynamoDB recibe todos los pares atributo-valor de la carga de acción que especifique. Para ver una lista de las regiones admitidas, consulte [Puntos de conexión y cuotas de Amazon DynamoDB](#) en la Referencia general de Amazon Web Services.

Amazon DynamoDB es un servicio de base de datos NoSQL totalmente administrado que ofrece un rendimiento rápido y predecible, así como una perfecta escalabilidad. Para obtener más información, consulte [¿Qué es DynamoDB?](#) en la Guía para desarrolladores de Amazon DynamoDB.

More information (10)

Al enviar datos a una columna de una tabla de DynamoDB, debe especificar los siguientes parámetros.

tableName

El nombre de la tabla de DynamoDB que recibe los datos. El valor `tableName` debe coincidir con el nombre de tabla de la tabla de DynamoDB. También puede especificar una expresión.

hashKeyField

El nombre de la clave hash (también denominada clave de partición). El valor `hashKeyField` debe coincidir con la clave de partición de la tabla de DynamoDB. También puede especificar una expresión.

hashKeyType

(Opcional) El tipo de datos de la clave hash. El valor del tipo de clave hash debe ser `STRING` o `NUMBER`. El valor predeterminado es `STRING`. También puede especificar una expresión.

hashKeyValue

El valor de la clave hash. La `hashKeyValue` utiliza plantillas de sustitución. Estas plantillas proporcionan datos en tiempo de ejecución. También puede especificar una expresión.

rangeKeyField

(Opcional) El nombre de la clave de rango (también denominada clave de clasificación). El valor `rangeKeyField` debe coincidir con la clave de clasificación de la tabla de DynamoDB. También puede especificar una expresión.

rangeKeyType

(Opcional) El tipo de datos de la clave de rango. El valor del tipo de clave hash debe ser `STRING` o `NUMBER`. El valor predeterminado es `STRING`. También puede especificar una expresión.

rangeKeyValue

(Opcional) El valor de la clave de rango. La `rangeKeyValue` utiliza plantillas de sustitución. Estas plantillas proporcionan datos en tiempo de ejecución. También puede especificar una expresión.

operación

(Opcional) El tipo de operación que se va a realizar. También puede especificar una expresión. El valor de la operación debe ser uno de los siguientes:

- `INSERT`: permite insertar datos como un elemento nuevo en la tabla de DynamoDB. Este es el valor predeterminado.
- `UPDATE`: permite actualizar un elemento existente de la tabla de DynamoDB con nuevos datos.
- `DELETE`: permite eliminar un elemento existente de la tabla de DynamoDB.

payloadField

(Opcional) El nombre de la columna de DynamoDB que recibe la carga de acción. El nombre predeterminado es `payload`. También puede especificar una expresión.

payload

(Opcional) La carga predeterminada contiene todos los pares atributo-valor que tienen la información sobre la instancia del modelo detector y el evento que desencadenó la acción. También puede personalizar la carga. Para obtener más información consulte [Carga](#) en la Referencia de la API de AWS IoT Events .

Si el tipo de carga especificado es una cadena, DynamoDBAction envía datos no JSON a la tabla de DynamoDB como datos binarios. La consola de DynamoDB mostrará los datos como texto codificado en Base64. El valor de payloadField es *payload-field_raw*. También puede especificar una expresión.

Note

Asegúrese de que la política adjunta a su función AWS IoT Events de servicio conceda el `dynamodb:PutItem` permiso. Para obtener más información, consulte [Gestión de identidad y acceso para AWS IoT Events](#).

Para obtener más información, consulte [Dynamo DBAction](#) en la referencia de la AWS IoT Events API.

Amazon DynamoDB(v2)

DynamoDBv2 action

La acción Amazon DynamoDB(v2) le permite escribir datos en una tabla de DynamoDB. Una columna independiente de la tabla de DynamoDB recibe un par atributo-valor de la carga de acción que especifique. Para ver una lista de las regiones admitidas, consulte [Puntos de conexión y cuotas de Amazon DynamoDB](#) en la Referencia general de Amazon Web Services.

Amazon DynamoDB es un servicio de base de datos NoSQL totalmente administrado que ofrece un rendimiento rápido y predecible, así como una perfecta escalabilidad. Para obtener más información, consulte [¿Qué es DynamoDB?](#) en la Guía para desarrolladores de Amazon DynamoDB.

More information (2)

Al enviar datos a varias columnas de una tabla de DynamoDB, debe especificar los siguientes parámetros.

tableName

El nombre de la tabla de DynamoDB que recibe los datos. También puede especificar una expresión.

payload

(Opcional) La carga predeterminada contiene todos los pares atributo-valor que tienen la información sobre la instancia del modelo detector y el evento que desencadenó la acción. También puede personalizar la carga. Para obtener más información consulte [Carga](#) en la Referencia de la API de AWS IoT Events .

Important

El tipo de carga debe ser JSON. También puede especificar una expresión.

Note

Asegúrese de que la política adjunta a su función de AWS IoT Events servicio conceda el `dynamodb:PutItem` permiso. Para obtener más información, consulte [Gestión de identidad y acceso para AWS IoT Events](#).

Para obtener más información, consulte [Dynamo DBv2 Action](#) en la referencia de la AWS IoT Events API.

Amazon Data Firehose

Firehose action

La acción Amazon Data Firehose te permite enviar datos a una cadena de entrega de Firehose. Para ver la lista de regiones compatibles, consulte los [puntos de enlace y las cuotas de Amazon Data Firehose](#) en. Referencia general de Amazon Web Services

Amazon Data Firehose es un servicio totalmente gestionado para entregar datos de streaming en tiempo real a destinos como Amazon Simple Storage Service (Amazon Simple Storage Service), Amazon Redshift, OpenSearch Amazon OpenSearch Service (Service) y Splunk. Para obtener más información, consulte [¿Qué es Amazon Data Firehose?](#) en la Guía para desarrolladores de Amazon Data Firehose.

More information (3)

Al enviar datos a un flujo de entrega de Firehose, debe especificar los siguientes parámetros.

deliveryStreamName

El nombre del flujo de entrega de Firehose que recibe los datos.

separator

(Opcional) Puede usar un separador de caracteres para separar los datos continuos enviados al flujo de entrega de Firehose. El valor del separador debe ser '`\n`' (nueva línea), '`\t`' (tabulador), '`\r\n`' (nueva línea de Windows) o '`,`' (coma).

payload

(Opcional) La carga predeterminada contiene todos los pares atributo-valor que tienen la información sobre la instancia del modelo detector y el evento que desencadenó la acción. También puede personalizar la carga. Para obtener más información consulte [Carga](#) en la Referencia de la API de AWS IoT Events .

Note

Asegúrese de que la política asociada a su función de AWS IoT Events servicio conceda el `firehose:PutRecord` permiso. Para obtener más información, consulte [Gestión de identidad y acceso para AWS IoT Events](#).

Para obtener más información, consulta [FirehoseAction](#) en la AWS IoT Events Referencia de la API de .

AWS Lambda

Lambda action

La AWS Lambda acción permite llamar a una función Lambda. Para ver una lista completa de las regiones compatibles, consulte [AWS Lambda endpoints and quotas](#) en la Referencia general de Amazon Web Services.

AWS Lambda es un servicio informático que permite ejecutar código sin aprovisionar ni administrar servidores. Para obtener más información, consulta [¿Qué es? AWS Lambda](#) en la Guía para AWS Lambda desarrolladores.

More information (2)

Al llamar a una función de Lambda, debe especificar los siguientes parámetros.

functionArn

El ARN de la función de Lambda a la que llamar.

payload

(Opcional) La carga predeterminada contiene todos los pares atributo-valor que tienen la información sobre la instancia del modelo detector y el evento que desencadenó la acción. También puede personalizar la carga. Para obtener más información consulte [Carga](#) en la Referencia de la API de AWS IoT Events .

Note

Asegúrese de que la política asociada a su función de AWS IoT Events servicio le conceda el `lambda:InvokeFunction` permiso. Para obtener más información, consulte [Gestión de identidad y acceso para AWS IoT Events](#).

Para obtener más información, consulta [LambdaAction](#) en la AWS IoT Events Referencia de la API de .

Amazon Simple Notification Service

SNS action

La acción de publicación de temas de Amazon SNS le permite publicar un mensaje de Amazon SNS. Para obtener la lista de las regiones admitidas, consulte [Puntos de conexión y cuotas de Amazon Simple Notification Service](#) en la Referencia general de Amazon Web Services.

Amazon Simple Notification Service (Amazon Simple Notification Service) es un servicio web que coordina y gestiona la entrega o el envío de mensajes a los puntos de conexión o clientes suscritos. Para obtener más información, consulte [¿Qué es Amazon SNS?](#) en la Guía para desarrolladores de Amazon Simple Notification Service.

Note

La acción de publicación de temas de Amazon SNS no admite temas FIFO (primero en entrar, primero en salir) de Amazon SNS. Dado que el motor de reglas es un servicio totalmente distribuido, es posible que los mensajes no se muestren en un orden específico al iniciar la acción de Amazon SNS.

More information (2)

Al publicar un mensaje de Amazon SNS, debe especificar los siguientes parámetros.

targetArn

El ARN del destino de Amazon SNS que recibe el mensaje.

payload

(Opcional) La carga predeterminada contiene todos los pares atributo-valor que tienen la información sobre la instancia del modelo detector y el evento que desencadenó la acción. También puede personalizar la carga. Para obtener más información consulte [Carga](#) en la Referencia de la API de AWS IoT Events .

Note

Asegúrese de que la política adjunta a su función AWS IoT Events de servicio conceda el `sns:Publish` permiso. Para obtener más información, consulte [Gestión de identidad y acceso para AWS IoT Events](#).

Para obtener más información, consulta [SNSTopicPublishAction](#) en la AWS IoT Events Referencia de la API de .

Amazon Simple Queue Service

SQS action

La acción Amazon SQS le permite enviar datos a una cola de Amazon SQS. Para obtener una lista de las regiones admitidas, consulte [Puntos de conexión y cuotas de Amazon Simple Queue Service](#) en la Referencia general de Amazon Web Services.

Amazon Simple Queue Service (Amazon SQS) ofrece una cola alojada segura, duradera y disponible que le permite integrar y desacoplar sistemas y componentes de software distribuidos. Para obtener más información, consulte [Qué es Amazon Simple Queue Service](#) en la Guía para desarrolladores de Amazon Simple Queue Service.

Note

La acción Amazon SQS no admite los temas >Amazon SQS FIFO (primero en entrar, primero en salir). Dado que el motor de reglas es un servicio totalmente distribuido, es posible que los mensajes no se muestren en un orden específico al iniciar la acción de Amazon SQS.

More information (3)

Al enviar datos a una cola de Amazon SQS, debe especificar los siguientes parámetros.

queueUrl

La URL de la cola de Amazon SQS que recibe los datos.

useBase64

(Opcional) AWS IoT Events codifica los datos en texto en Base64, si lo especifica. TRUE El valor predeterminado es FALSE.

payload

(Opcional) La carga predeterminada contiene todos los pares atributo-valor que tienen la información sobre la instancia del modelo detector y el evento que desencadenó la acción. También puede personalizar la carga. Para obtener más información consulte [Carga](#) en la Referencia de la API de AWS IoT Events .

Note

Asegúrese de que la política adjunta a su función de AWS IoT Events servicio conceda el `sqs:SendMessage` permiso. Para obtener más información, consulte [Gestión de identidad y acceso para AWS IoT Events](#).

Para obtener más información, consulta [SNSTopicPublishAction](#) en la AWS IoT Events Referencia de la API de .

También puede utilizar Amazon SNS y el motor de AWS IoT Core reglas para activar una AWS Lambda función. Esto permite realizar acciones utilizando otros servicios, como Amazon Connect, o incluso una aplicación de planificación de recursos empresariales (ERP) de la empresa.

Note

Para recopilar y procesar grandes flujos de registros de datos en tiempo real, puede utilizar otros AWS servicios, como [Amazon Kinesis](#). Desde allí, puede completar un análisis inicial y, a continuación, enviar los resultados AWS IoT Events como entrada a un detector.

Expresiones para filtrar, transformar y procesar datos de eventos

Las expresiones se utilizan para evaluar los datos entrantes, realizar cálculos y determinar las condiciones en las que deben producirse acciones o transiciones de estado específicas. AWS IoT Events proporciona varias formas de especificar valores al crear y actualizar modelos de detectores. Puede usar expresiones para especificar valores literales o AWS IoT Events puede evaluar las expresiones antes de especificar valores determinados.

Temas

- [Sintaxis para filtrar los datos del dispositivo y definir acciones en AWS IoT Events](#)
- [Ejemplos de expresiones y uso para AWS IoT Events](#)

Sintaxis para filtrar los datos del dispositivo y definir acciones en AWS IoT Events

Las expresiones ofrecen una sintaxis para filtrar los datos del dispositivo y definir acciones. Puede utilizar literales, operadores, funciones, referencias y plantillas de sustitución en las expresiones de AWS IoT Events . Al combinar estos componentes, puede crear expresiones potentes y flexibles para procesar datos de IoT, realizar cálculos, manipular cadenas y tomar decisiones lógicas dentro de sus modelos de detectores.

Literales

- Entero
- Decimal
- Cadena
- Booleano

Operadores

Unario

- Not (Booleano): !

- Not (bit a bit): ~
- Menos (aritmético): -

Cadena

- Concatenación: +

Ambos operandos deben ser cadenas. Los literales de cadena se deben incluir entre comillas simples (').

Por ejemplo: 'my' + 'string' -> 'mystring'

Aritmético

- Adición: +

Ambos operandos deben ser numéricos.

- Sustracción: -
- División: /

El resultado de la división es un valor entero redondeado, a menos que al menos uno de los operandos (divisor o dividendo) sea un valor decimal.

- Multiplicación: *

Bit a bit (entero)

- O BIEN: |

Por ejemplo: 13 | 5 -> 13

- Y: &

Por ejemplo: 13 & 5 -> 5

- XOR: ^

Por ejemplo: 13 ^ 5 -> 8

- NOT: ~

Por ejemplo: ~13 -> -14

Booleano

- Menor que: <
- Menor o igual que: <=

- Igual a: `==`
- No igual a: `!=`
- Mayor o igual que: `>=`
- Mayor que: `>`
- Y: `&&`
- O BIEN: `||`

**Note**

Cuando una subexpresión de `||` contiene datos indefinidos, esa subexpresión se trata como `false`.

Paréntesis

Puede usar paréntesis para agrupar términos dentro de una expresión.

Funciones para usar en expresiones AWS IoT Events

AWS IoT Events proporciona un conjunto de funciones integradas para mejorar las capacidades de las expresiones del modelo de su detector. Estas funciones permiten la gestión del temporizador, la conversión de tipos, la comprobación de valores nulos, la identificación del tipo de disparador, la verificación de las entradas, la manipulación de cadenas y las operaciones bit a bit. Al aprovechar estas funciones, puede crear una lógica de AWS IoT Events procesamiento receptiva, lo que mejora la eficacia general de sus aplicaciones de IoT.

Funciones integradas

`timeout("timer-name")`

Da como resultado `true` si ha transcurrido el tiempo especificado. Sustituya `timer-name` por el nombre del temporizador que haya definido, entre comillas. En una acción de evento, puede definir un temporizador y luego iniciarlo, restablecerlo o borrar uno que haya definido con anterioridad. Consulte el campo `detectorModelDefinition.states.onInput|onEnter|onExit.events.actions.setTimer.timerName`.

Un temporizador creado en un estado puede ser referenciado en un estado diferente. Debe visitar el estado en el que creó el temporizador antes de entrar en el estado en el que se hace referencia al temporizador.

Por ejemplo, un modelo de detector tiene dos estados: `TemperatureChecked` y `RecordUpdated`. Ha creado un temporizador en el `TemperatureChecked` estado. Primero debe visitar el `TemperatureChecked` estado antes de poder usar el temporizador en el `RecordUpdated` estado.

Para garantizar la precisión, el tiempo mínimo que debe fijarse un temporizador es de 60 segundos.

 Note

`timeout()` devuelve `true` solo la primera vez que se comprueba tras el vencimiento real del temporizador y devuelve `false` a partir de entonces.

convert(*type*, *expression*)

Da como resultado el valor de la expresión convertida al tipo especificado. El *type* valor debe ser `String`, `Boolean`, o `Decimal`. Utilice una de estas palabras clave o una expresión que dé como resultado una cadena que contenga la palabra clave. Solo las siguientes conversiones tienen éxito y devuelven un valor válido:

- Booleano -> cadena

Devuelve la cadena `"true"` o `"false"`.

- Decimal -> cadena
- Cadena -> Booleano
- Cadena -> decimal

La cadena especificada debe ser una representación válida de un número decimal o `convert()` falla.

Si `convert()` no devuelve un valor válido, la expresión de la que forma parte tampoco es válida. Este resultado equivale a `false` y no activará las `actions` o la transición al `nextState` especificado como parte del evento en el que se produce la expresión.

isNull(*expression*)

Da como resultado `true` si la expresión devuelve un valor nulo. Por ejemplo, si la entrada `MyInput` recibe el mensaje `{ "a": null }`, entonces lo siguiente da como resultado `true`, pero `isUndefined($input.MyInput.a)` da como resultado `false`.

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

isUndefined(*expression*)

Da como resultado `true` si la expresión es indefinida. Por ejemplo, si la entrada `MyInput` recibe el mensaje `{ "a": null }`, entonces lo siguiente da como resultado `false`, pero `isNull($input.MyInput.a)` da como resultado `true`.

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

triggerType("type")

El valor de *type* puede ser `"Message"` o `"Timer"`. Se evalúa como `true` si la condición del evento en la que aparece se está evaluando porque un temporizador se ha vencido, como en el siguiente ejemplo.

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

O se ha recibido un mensaje de entrada.

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

currentInput("input")

Da como resultado `true` si la condición del evento en la que aparece se está evaluando porque se ha recibido el mensaje de entrada especificado. Por ejemplo, si la entrada `Command` recibe el mensaje `{ "value": "Abort" }`, entonces da como resultado `true`.

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

Utilice esta función para verificar que la condición se está evaluando porque se ha recibido un mensaje de entrada concreto y no se ha vencido un temporizador, como en la siguiente expresión.

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

Funciones de coincidencia de cadenas

startsWith(*expression1*, *expression2*)

Da como resultado `true` si la primera expresión de cadena comienza con la segunda expresión de cadena. Por ejemplo, si la entrada `MyInput` recibe el mensaje `{ "status": "offline" }`, entonces lo siguiente da como resultado `true`.

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

Ambas expresiones deben dar como resultado un valor de cadena. Si alguna de las expresiones no da como resultado un valor de cadena, entonces el resultado de la función es indefinido. No se realiza ninguna conversión.

endsWith(*expression1*, *expression2*)

Da como resultado `true` si la primera expresión de cadena termina con la segunda expresión de cadena. Por ejemplo, si la entrada `MyInput` recibe el mensaje `{ "status": "offline" }`, entonces lo siguiente da como resultado `true`.

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

Ambas expresiones deben dar como resultado un valor de cadena. Si alguna de las expresiones no da como resultado un valor de cadena, entonces el resultado de la función es indefinido. No se realiza ninguna conversión.

contains(*expression1*, *expression2*)

Da como resultado `true` si la primera expresión de cadena contiene la segunda expresión de cadena. Por ejemplo, si la entrada `MyInput` recibe el mensaje `{ "status": "offline" }`, entonces lo siguiente da como resultado `true`.

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

Ambas expresiones deben dar como resultado un valor de cadena. Si alguna de las expresiones no da como resultado un valor de cadena, entonces el resultado de la función es indefinido. No se realiza ninguna conversión.

Funciones de manipulación de enteros bit a bit

bitor(*expression1*, *expression2*)

Evalúa el OR bit a bit de las expresiones de enteros (la operación OR binaria se realiza en los bits correspondientes de los enteros). Por ejemplo, si la entrada MyInput recibe el mensaje { "value1": 13, "value2": 5 }, entonces lo siguiente da como resultado 13.

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

Ambas expresiones deben dar como resultado un valor entero. Si cualquiera de las expresiones no da como resultado un valor entero, el resultado de la función es indefinido. No se realiza ninguna conversión.

bitand(*expression1*, *expression2*)

Evalúa el AND bit a bit de las expresiones de enteros (la operación AND binaria se realiza en los bits correspondientes de los enteros). Por ejemplo, si la entrada MyInput recibe el mensaje { "value1": 13, "value2": 5 }, entonces lo siguiente da como resultado 5.

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

Ambas expresiones deben dar como resultado un valor entero. Si cualquiera de las expresiones no da como resultado un valor entero, el resultado de la función es indefinido. No se realiza ninguna conversión.

bitxor(*expression1*, *expression2*)

Evalúa el XOR bit a bit de las expresiones de enteros (la operación XOR binaria se realiza en los bits correspondientes de los enteros). Por ejemplo, si la entrada MyInput recibe el mensaje { "value1": 13, "value2": 5 }, entonces lo siguiente da como resultado 8.

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

Ambas expresiones deben dar como resultado un valor entero. Si cualquiera de las expresiones no da como resultado un valor entero, el resultado de la función es indefinido. No se realiza ninguna conversión.

bitnot(*expression*)

Evalúa el NOT bit a bit de la expresión de enteros (la operación NOT binaria se realiza en los bits del entero). Por ejemplo, si la entrada MyInput recibe el mensaje { "value": 13 }, entonces lo siguiente da como resultado -14.

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

Ambas expresiones deben dar como resultado un valor entero. Si cualquiera de las expresiones no da como resultado un valor entero, el resultado de la función es indefinido. No se realiza ninguna conversión.

AWS IoT Events referencia para las entradas y variables en las expresiones

Entradas

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

`input-name` es una entrada que se crea mediante la [CreateInput](#) acción.

Por ejemplo, si tiene una entrada llamada TemperatureInput para la que ha definido entradas de `inputDefinition.attributes.jsonPath`, los valores podrían aparecer en los siguientes campos disponibles.

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

Para hacer referencia al valor del campo `temperature`, utilice el siguiente comando.

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

Para los campos cuyos valores son matrices, puede hacer referencia a los miembros de la matriz mediante `[n]`. Por ejemplo, dados los siguientes valores:

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

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

Se puede hacer referencia al valor 78.8 con el siguiente comando.

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

Variables

`$variable.variable-name`

variable-name Es una variable que se definió mediante la [CreateDetectorModel](#) acción.

Por ejemplo, si tiene una variable llamada TechnicianID que ha definido mediante `detectorDefinition.states.onInputEvents.actions.setVariable.variableName`, puede hacer referencia al valor (cadena) dado más recientemente a la variable con el siguiente comando.

```
$variable.TechnicianID
```

Solo puede establecer los valores de las variables mediante la acción `setVariable`. No puede asignar valores a variables en una expresión. Una variable no se puede destruir. Por ejemplo, no puede asignarle el valor `null`.

Note

En las referencias que utilicen identificadores que no sigan el patrón (expresión regular) `[a-zA-Z][a-zA-Z0-9_]*`, debe encerrar dichos identificadores entre tildes graves (```). Por ejemplo, una referencia a una entrada llamada MyInput con un campo llamado `_value` debe especificar este campo como `$input.MyInput.`_value``.

Cuando utilice referencias en expresiones, compruebe lo siguiente:

- Cuando utilice una referencia como operando con uno o más operadores, asegúrese de que todos los tipos de datos a los que hace referencia sean compatibles.

Por ejemplo, en la siguiente expresión, el entero 2 es un operando de los operadores `==` y `&&`. Para asegurarse de que los operandos sean compatibles, `$variable.testVariable + 1` y `$variable.testVariable` deben hacer referencia a un número entero o decimal.

Además, el entero 1 es un operando del operador `+`. Por lo tanto, `$variable.testVariable` debe hacer referencia a un número entero o decimal.

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

- Cuando utilice una referencia como argumento pasado a una función, asegúrese de que la función admita los tipos de datos a los que hace referencia.

Por ejemplo, la siguiente función `timeout("time-name")` requiere una cadena con comillas dobles como argumento. Si utiliza una referencia para el `timer-name` valor, debe hacer referencia a una cadena con comillas dobles.

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

Note

Para la `convert(type, expression)` función, si utiliza una referencia para el `type` valor, el resultado evaluado de la referencia debe ser `StringDecimal`, o `Boolean`.

AWS IoT Events las expresiones admiten tipos de datos enteros, decimales, de cadena y booleanos. En la tabla siguiente se ofrece una lista de pares de tipos incompatibles.

Pares de tipos incompatibles

Entero, cadena

Entero, booleano

Decimal, cadena

Decimal, booleano

Cadena, booleano

Plantillas de sustitución para expresiones AWS IoT Events

'\${*expression*}'

`${}` identifica la cadena como una cadena interpolada. *expression* Puede ser cualquier AWS IoT Events expresión. Esto incluye operadores, funciones y referencias.

Por ejemplo, ha utilizado la [SetVariableAction](#) acción para definir una variable. El `variableName` es `SensorID` y el `value` es `10`. Puede crear las siguientes plantillas de sustitución.

Plantilla de sustitución	Cadena de resultados
'\${'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}"

Ejemplos de expresiones y uso para AWS IoT Events

Puede especificar valores en un modelo de detector de las siguientes maneras:

- Introduzca las expresiones compatibles en la AWS IoT Events consola.
- Pase las expresiones a los parámetros AWS IoT Events APIs as.

Las expresiones admiten literales, operadores, funciones, referencias y plantillas de sustitución.

 Important

Sus expresiones deben hacer referencia a un valor entero, decimal, de cadena o booleano.

Escribir AWS IoT Events expresiones

Consulta los siguientes ejemplos para ayudarte a escribir tus AWS IoT Events expresiones:

Literal

En los valores literales, las expresiones deben contener comillas simples. Un valor booleano debe ser `true` o `false`.

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

Referencia

En las referencias, debe especificar variables o valores de entrada.

- La siguiente entrada hace referencia a un número decimal, `10.01`.

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

- La siguiente variable hace referencia a una cadena, `Greenhouse Temperature Table`.

```
$variable.TableName
```

Plantilla de sustitución

En una plantilla de sustitución, debe usar `${}` y la plantilla debe estar entre comillas simples. Una plantilla de sustitución también puede contener una combinación de literales, operadores, funciones, referencias y plantillas de sustitución.

- El resultado evaluado de la siguiente expresión es una cadena, `50.018 in Fahrenheit`.

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

- El resultado evaluado de la siguiente expresión es una cadena, `{"sensor_id\":"Sensor_1\","temperature\":"50.018\}"`.

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

Concatenación de cadenas

En una concatenación de cadenas, debe usar `+`. Una concatenación de cadenas también puede contener una combinación de literales, operadores, funciones, referencias y plantillas de sustitución.

- El resultado evaluado de la siguiente expresión es una cadena, `Greenhouse Temperature Table 2000-01-01`.

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

AWS IoT Events ejemplos de modelos de detectores

Esta página proporciona una lista de ejemplos de casos de uso que muestran cómo configurar varias AWS IoT Events funciones. Los ejemplos van desde detecciones básicas, como los umbrales de temperatura, hasta escenarios más avanzados de detección de anomalías y aprendizaje automático. Cada ejemplo incluye procedimientos y fragmentos de código para ayudarle a configurar las AWS IoT Events detecciones, las acciones y las integraciones. Estos ejemplos muestran la flexibilidad del AWS IoT Events servicio y cómo se puede personalizar para diversas aplicaciones y casos de uso de IoT. Consulte esta página cuando explore AWS IoT Events las capacidades o si necesita orientación para implementar un flujo de trabajo específico de detección o automatización.

Temas

- [Ejemplo: usar el control de temperatura de HVAC con AWS IoT Events](#)
- [Ejemplo: una grúa que detecta las condiciones mediante AWS IoT Events](#)
- [Envíe comandos en respuesta a las condiciones detectadas en AWS IoT Events](#)
- [Un modelo AWS IoT Events de detector para el monitoreo de grúas](#)
- [AWS IoT Events entradas para el monitoreo de grúas](#)
- [Envíe mensajes de alarma y operativos con AWS IoT Events](#)
- [Ejemplo: detección de AWS IoT Events eventos con sensores y aplicaciones](#)
- [Ejemplo: dispositivo HeartBeat con el que monitorizar las conexiones del dispositivo AWS IoT Events](#)
- [Ejemplo: una alarma ISA en AWS IoT Events](#)
- [Ejemplo: cree una alarma sencilla con AWS IoT Events](#)

Ejemplo: usar el control de temperatura de HVAC con AWS IoT Events

Antecedentes

Este ejemplo implementa un modelo de control de temperatura (un termostato) con estas características:

- Un modelo de detector que usted define que puede supervisar y controlar múltiples áreas. (Se creará una instancia de detector para cada área).

- Cada instancia de detector recibe datos de temperatura de múltiples sensores dispuestos en cada área de control.
- Puede cambiar la temperatura deseada (el punto de consigna) para cada área en cualquier momento.
- Puede definir los parámetros operativos para cada área y cambiar estos parámetros en cualquier momento.
- Puede añadir sensores a un área o eliminarlos de ella en cualquier momento.
- Puede activar un tiempo mínimo de funcionamiento de las unidades de calefacción y refrigeración para protegerlas de posibles daños.
- Los detectores rechazarán y reportarán las lecturas anómalas de los sensores.
- Puede definir puntos de consigna de temperatura de emergencia. Si cualquier sensor reporta una temperatura superior o inferior a los puntos de consigna que haya definido, las unidades de calefacción o refrigeración se activarán de inmediato y el detector reportará ese pico de temperatura.

Este ejemplo demuestra las siguientes capacidades funcionales:

- Creación de modelos de detectores de eventos.
- Creación de entradas.
- Ingesta de entradas en un modelo de detector.
- Evaluación de las condiciones de activación.
- Consulta de las variables de estado en las condiciones y ajuste de los valores de las variables en función de las condiciones.
- Consulta de los temporizadores en las condiciones y ajuste de los temporizadores en función de las condiciones.
- Medidas de acción que envíen mensajes MQTT y Amazon SNS.

Introduzca las definiciones de un sistema de calefacción, ventilación y aire acondicionado en AWS IoT Events

Se utiliza una `seedTemperatureInput` para crear una instancia de detector para un área y definir sus parámetros operativos.

La configuración de las entradas de los sistemas de climatización AWS IoT Events es importante para un control climático eficaz. En este ejemplo, se muestra cómo configurar entradas que capturen datos como la temperatura, la humedad, la ocupación y el consumo de energía. Aprenda a definir los atributos de entrada, configurar las fuentes de datos y configurar las reglas de preprocesamiento para ayudar a los modelos de sus detectores a recibir información precisa y oportuna para una gestión y eficiencia óptimas.

Comando CLI utilizado:

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

Archivo: 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" }
    ]
  }
}
```

Respuesta:

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

Cada sensor de cada área debe enviar una `temperatureInput`, según sea necesario.

Comando CLI utilizado:

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

Archivo: `temperatureInput.json`

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

Respuesta:

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

Definición del modelo de detector para un sistema de climatización mediante AWS IoT Events

El `areaDetectorModel` define cómo funciona cada instancia de detector. Cada instancia de `state machine` recoge las lecturas de los sensores de temperatura, luego cambia el estado y envía mensajes de control en función de estas lecturas.

Comando CLI utilizado:

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

Archivo: `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 <=
$variable.anomalousLow",
    "actions": [
      {
        "iotTopicPublish": {
          "mqttTopic": "temperatureSensor/anomaly"
        }
      }
    ]
  },
  {
    "eventName": "highTemperatureSpike",
    "condition": "$input.temperatureInput.sensorData.temperature >
$variable.rangeHigh",
    "actions": [
      {
        "iotTopicPublish": {
          "mqttTopic": "temperatureSensor/spike"
        }
      }
    ]
  }
]
}

```

```

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

{
  "eventName": "lowTemperatureSpike",
  "condition": "$input.temperatureInput.sensorData.temperature <
$variable.rangeLow",
  "actions": [
    {
      "iotTopicPublish": {
        "mqttTopic": "temperatureSensor/spike"
      }
    },
    {
      "sns": {
        "targetArn": "arn:aws:sns:us-west-2:123456789012:cool0ff"
      }
    },
    {
      "sns": {
        "targetArn": "arn:aws:sns:us-west-2:123456789012:heat0n"
      }
    },
    {
      "iotTopicPublish": {
        "mqttTopic": "hvac/Cooling/Off"
      }
    },
    {
      "iotTopicPublish": {
        "mqttTopic": "hvac/Heating/On"
      }
    },
    {
      "setVariable": {
        "variableName": "enteringNewState",
        "value": "true"
      }
    }
  ],
  "nextState": "heating"
},

```

```

    {
      "eventName": "desiredTemperature",
      "condition": "((((($variable.averageTemperature * ($variable.sensorCount
- 1)) + $input.temperatureInput.sensorData.temperature) / $variable.sensorCount) <=
($variable.desiredTemperature - $variable.allowedError)) && $variable.goodToGo ==
true",
      "actions": [
        {
          "sns": {
            "targetArn": "arn:aws:sns:us-west-2:123456789012:cool0ff"
          }
        },
        {
          "iotTopicPublish": {
            "mqttTopic": "hvac/Cooling/Off"
          }
        }
      ],
      "nextState": "idle"
    }
  ]
},
{
  "stateName": "heating",
  "onEnter": {
    "events": [
      {
        "eventName": "delay",
        "condition": "(!$variable.noDelay && $variable.enteringNewState",
        "actions": [
          {
            "setTimer": {
              "timerName": "heatingTimer",
              "seconds": 120
            }
          },
          {
            "setVariable": {
              "variableName": "goodToGo",
              "value": "false"
            }
          }
        ]
      }
    ]
  }
}

```

```

        }
      }
    ]
  },
  {
    "eventName": "dontDelay",
    "condition": "$variable.noDelay == true",
    "actions": [
      {
        "setVariable": {
          "variableName": "goodToGo",
          "value": "true"
        }
      }
    ]
  },
  {
    "eventName": "beenHere",
    "condition": "true",
    "actions": [
      {
        "setVariable": {
          "variableName": "enteringNewState",
          "value": "false"
        }
      }
    ]
  }
]
},
"onInput": {
  "events": [
    {
      "eventName": "whatWasInput",
      "condition": "true",
      "actions": [
        {
          "setVariable": {
            "variableName": "sensorId",
            "value": "$input.temperatureInput.sensorId"
          }
        }
      ],
    },
    {

```

```

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

```

```

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

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

```

```

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

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

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

```

```

        "mqttTopic": "hvac/Heating/Off"
      }
    }
  ],
  "nextState": "idle"
}
]
}
},
"initialStateName": "start"
},
"key": "areaId",
"roleArn": "arn:aws:iam::123456789012:role/IoTEventsRole"
}

```

Respuesta:

```

{
  "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 ejemplos de un sistema de calefacción, ventilación y aire acondicionado en AWS IoT Events

En este ejemplo, BatchPutMessage se utiliza para crear una instancia de detector para un área y definir los parámetros de funcionamiento iniciales.

Comando CLI utilizado:

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

Archivo: 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}"
    }
  ]
}
```

Respuesta:

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

En este ejemplo, BatchPutMessage se utiliza para reportar las lecturas de los sensores de temperatura, de un único sensor en un área.

Comando CLI utilizado:

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

Archivo: temperatureExample.json

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

```
]
}
```

Respuesta:

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

En este ejemplo, BatchPutMessage se utiliza para cambiar la temperatura deseada de un área.

Comando CLI utilizado:

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

Archivo: seedSetDesiredTemp.json

```
{
  "messages": [
    {
      "messageId": "000001",
      "inputName": "seedTemperatureInput",
      "payload": "{\"areaId\": \"Area51\", \"desiredTemperature\": 23.0}"
    }
  ]
}
```

Respuesta:

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

Ejemplos de mensajes Amazon SNS generados por la instancia de detector del Area51:

```
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"
}

```

En este ejemplo, utilizamos la API `DescribeDetector` para obtener información sobre el estado actual de una instancia de detector.

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

Respuesta:

```

{
  "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 ejemplo de un sistema de climatización en AWS IoT Events

En este ejemplo, BatchUpdateDetector se utiliza para cambiar los parámetros operativos de una instancia de detector en funcionamiento.

La administración eficiente del sistema de HVAC a menudo requiere actualizaciones por lotes de varios detectores. En esta sección se muestra cómo utilizar AWS IoT Events la función de actualización por lotes para los detectores. Aprenda a modificar simultáneamente varios parámetros de control y a actualizar los valores límite para poder ajustar las acciones de respuesta en una flota de dispositivos, lo que mejorará su capacidad de gestionar sistemas a gran escala de forma eficaz.

Comando CLI utilizado:

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

Archivo: 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": [
    ]
  }
}
```

Respuesta:

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

El motor de AWS IoT Core reglas y AWS IoT Events

Las siguientes reglas vuelven a publicar los mensajes AWS IoT Events MQTT como mensajes de solicitud de actualización paralelos. Suponemos que AWS IoT Core las cosas están definidas para

una unidad de calefacción y una unidad de refrigeración para cada área controlada por el modelo de detector.

En este ejemplo, hemos definido cosas denominadas `Area51HeatingUnit` y `Area51CoolingUnit`.

Comando CLI utilizado:

```
aws iot create-topic-rule --cli-input-json file://ADMSHadowCool0ffRule.json
```

Archivo: `ADMSHadowCool0ffRule.json`

```
{
  "ruleName": "ADMSHadowCool0ff",
  "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"
        }
      }
    ]
  }
}
```

Respuesta: [vacío]

Comando CLI utilizado:

```
aws iot create-topic-rule --cli-input-json file://ADMSHadowCool0nRule.json
```

Archivo: `ADMSHadowCool0nRule.json`

```
{
  "ruleName": "ADMSHadowCool0n",
```

```

"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"
      }
    }
  ]
}

```

Respuesta: [vacío]

Comando CLI utilizado:

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

Archivo: 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"
        }
      }
    ]
  }
}

```

```
}  
}
```

Respuesta: [vacío]

Comando CLI utilizado:

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

Archivo: 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"  
        }  
      }  
    ]  
  }  
}
```

Respuesta: [vacío]

Ejemplo: una grúa que detecta las condiciones mediante AWS IoT Events

Un operador de muchas grúas desea detectar cuándo las máquinas necesitan mantenimiento o sustitución y activar las notificaciones apropiadas. Cada grúa tiene un motor. Cada motor emite mensajes (entradas) con información sobre presión y temperatura. El operador quiere dos niveles de detectores de eventos:

- Un detector de eventos a nivel de grúa
- Un detector de eventos a nivel de motor

Mediante los mensajes procedentes de los motores (que contienen metadatos que incluyen `craneId` y `motorId`), el operador puede ejecutar ambos niveles de detectores de eventos utilizando el enrutamiento apropiado. Al cumplirse las condiciones de un evento, las notificaciones deben enviarse a los temas apropiados de Amazon SNS. El operador puede configurar los modelos de detectores para que no se emitan notificaciones duplicadas.

Este ejemplo demuestra las siguientes capacidades funcionales:

- Creación, lectura, actualización y eliminación (CRUD) de entradas.
- Creación, lectura, actualización y eliminación (CRUD) de modelos de detectores de eventos y diferentes versiones de detectores de eventos.
- Enrutamiento de una entrada a múltiples detectores de eventos.
- Ingesta de entradas en un modelo de detector.
- Evaluación de las condiciones de activación y de eventos de ciclo de vida.
- Capacidad para consultar variables de estado en las condiciones y establecer sus valores en función de las condiciones.
- Orquestación en tiempo de ejecución con definición, estado, evaluador de desencadenantes y ejecutor de acciones.
- Ejecución de acciones en `ActionsExecutor` con un objetivo SNS.

Envíe comandos en respuesta a las condiciones detectadas en AWS IoT Events

Esta página proporciona un ejemplo del uso de AWS IoT Events comandos para configurar entradas, crear modelos de detectores y enviar datos de sensores simulados. Los ejemplos muestran cómo aprovechar AWS IoT Events para monitorear equipos industriales como motores y grúas.

```
#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
```

Un modelo AWS IoT Events de detector para el monitoreo de grúas

Supervise sus flotas de equipos o dispositivos para detectar fallas o cambios en la operación y active acciones cuando ocurran dichos eventos. Defina modelos de detectores en JSON que especifican estados, reglas y acciones. Esto le permite monitorear entradas como la temperatura y la presión, rastrear las infracciones de los umbrales y enviar alertas. Los ejemplos muestran modelos de detectores para una grúa y un motor, que detectan problemas de sobrecalentamiento y Amazon SNS los notifica cuando se supera un umbral. Puede actualizar los modelos para refinar el comportamiento sin interrumpir la supervisión.

Archivo: craneDetectorModel.json

```
{
  "detectorModelName": "craneDetectorModel",
  "detectorModelDefinition": {
    "states": [
      {
        "stateName": "Running",
        "onEnter": {
          "events": [
            {
              "eventName": "init",
              "condition": "true",
              "actions": [
                {
                  "setVariable": {
                    "variableName": "craneThresholdBreach",
                    "value": "0"
                  }
                }
              ]
            }
          ]
        }
      ]
    },
    "onInput": {
      "events": [
```

```

        {
            "eventName": "Overheated",
            "condition": "$input.TemperatureInput.temperature > 35",
            "actions": [
                {
                    "setVariable": {
                        "variableName": "craneThresholdBreach",
                        "value": "$variable.craneThresholdBreach + 1"
                    }
                }
            ]
        },
        {
            "eventName": "Crane Threshold Breach",
            "condition": "$variable.craneThresholdBreach > 5",
            "actions": [
                {
                    "sns": {
                        "targetArn": "arn:aws:sns:us-
east-1:123456789012:CraneSNSTopic"
                    }
                }
            ]
        },
        {
            "eventName": "Underheated",
            "condition": "$input.TemperatureInput.temperature < 25",
            "actions": [
                {
                    "setVariable": {
                        "variableName": "craneThresholdBreach",
                        "value": "0"
                    }
                }
            ]
        }
    ]
},
"initialStateName": "Running"
},
"key": "craneid",
"roleArn": "arn:aws:iam::123456789012:role/columboSNSRole"

```

```
}
```

Para actualizar un modelo de detector existente. Archivo: `updateCraneDetectorModel.json`

```
{
  "detectorModelName": "craneDetectorModel",
  "detectorModelDefinition": {
    "states": [
      {
        "stateName": "Running",
        "onEnter": {
          "events": [
            {
              "eventName": "init",
              "condition": "true",
              "actions": [
                {
                  "setVariable": {
                    "variableName": "craneThresholdBreach",
                    "value": "0"
                  }
                },
                {
                  "setVariable": {
                    "variableName": "alarmRaised",
                    "value": "'false'"
                  }
                }
              ]
            }
          ]
        },
        "onInput": {
          "events": [
            {
              "eventName": "Overheated",
              "condition": "$input.TemperatureInput.temperature > 30",
              "actions": [
                {
                  "setVariable": {
                    "variableName": "craneThresholdBreach",
                    "value": "$variable.craneThresholdBreach + 1"
                  }
                }
              ]
            }
          ]
        }
      }
    ]
  }
}
```

```

        }
    ],
    },
    {
        "eventName": "Crane Threshold Breached",
        "condition": "$variable.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"
}

```

Archivo: motorDetectorModel.json

```

{
  "detectorModelName": "motorDetectorModel",
  "detectorModelDefinition": {
    "states": [
      {
        "stateName": "Running",
        "onEnter": {
          "events": [
            {
              "eventName": "init",
              "condition": "true",
              "actions": [
                {
                  "setVariable": {
                    "variableName": "motorThresholdBreach",
                    "value": "0"
                  }
                }
              ]
            }
          ],
          "onInput": {
            "events": [
              {
                "eventName": "Overheated And Overpressurized",
                "condition": "$input.PressureInput.pressure > 70 && $input.TemperatureInput.temperature > 30",
                "actions": [
                  {
                    "setVariable": {
                      "variableName": "motorThresholdBreach",
                      "value": "$variable.motorThresholdBreach + 1"
                    }
                  }
                ]
              }
            ],
            {
              "eventName": "Motor Threshold Breached",
              "condition": "$variable.motorThresholdBreach > 5",
              "actions": [
                {

```

```

        "sns": {
            "targetArn": "arn:aws:sns:us-
east-1:123456789012:MotorSNSTopic"
        }
    },
    ],
    "initialStateName": "Running"
},
{
    "key": "motorId",
    "roleArn": "arn:aws:iam::123456789012:role/columboSNSRole"
}

```

Para actualizar un modelo de detector existente. Archivo: `updateMotorDetectorModel.json`

```

{
    "detectorModelName": "motorDetectorModel",
    "detectorModelDefinition": {
        "states": [
            {
                "stateName": "Running",
                "onEnter": {
                    "events": [
                        {
                            "eventName": "init",
                            "condition": "true",
                            "actions": [
                                {
                                    "setVariable": {
                                        "variableName": "motorThresholdBreach",
                                        "value": "0"
                                    }
                                }
                            ]
                        }
                    ]
                },
                "onInput": {
                    "events": [

```

```

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

```

AWS IoT Events entradas para el monitoreo de grúas

En este ejemplo, demostramos cómo configurar las entradas para un sistema de monitoreo de grúas utilizando AWS IoT Events. Captura las entradas de presión y temperatura para ilustrar cómo estructurar las entradas para el monitoreo de equipos industriales complejos.

Archivo: `pressureInput.json`

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

Archivo: temperatureInput.json

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

Envíe mensajes de alarma y operativos con AWS IoT Events

El manejo efectivo de los mensajes es importante en los sistemas de monitoreo de grúas. En esta sección se muestra cómo configurarlo para procesar y responder AWS IoT Events a varios tipos de mensajes procedentes de los sensores de la grúa. La configuración de alarmas en función de un mensaje concreto puede ayudarle a analizar, filtrar y enrutar las actualizaciones de estado para activar las acciones adecuadas.

Archivo: highPressureMessage.json

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

```
    }  
  ]  
}
```

Archivo: highTemperatureMessage.json

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

Archivo: lowPressureMessage.json

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

Archivo: lowTemperatureMessage.json

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

Ejemplo: detección de AWS IoT Events eventos con sensores y aplicaciones

Este modelo de detector es una de las plantillas disponibles en la AWS IoT Events consola. Se incluye aquí para su comodidad.

Este ejemplo muestra AWS IoT Events la detección de eventos de una aplicación mediante datos de sensores. Muestra cómo se puede crear un modelo de detector que supervise eventos específicos para poder activar las acciones adecuadas. Puede crear múltiples entradas de sensor, definir condiciones de eventos complejos y configurar mecanismos de respuesta graduales.

```
{
  "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"
}
}

```

Ejemplo: dispositivo HeartBeat con el que monitorizar las conexiones del dispositivo AWS IoT Events

Este modelo de detector es una de las plantillas disponibles en la AWS IoT Events consola. Se incluye aquí para su comodidad.

El ejemplo del latido cardíaco defectuoso (DHB) ilustra cómo se AWS IoT Events puede utilizar en la monitorización de la asistencia sanitaria. En este ejemplo se muestra cómo se puede crear un modelo de detector que analice los datos de frecuencia cardíaca, detecte patrones irregulares y active las respuestas adecuadas. Aprenda a configurar entradas, definir umbrales y configurar alertas para posibles problemas cardíacos, demostrando su versatilidad en aplicaciones relacionadas con el cuidado AWS IoT Events de la salud.

```

{
  "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"
}
}

```

Ejemplo: una alarma ISA en AWS IoT Events

Este modelo de detector es una de las plantillas disponibles en la AWS IoT Events consola. Se incluye aquí para su comodidad.

```

{
    "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": "\\rtunack\\"
              }
            }
          ],
          "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"
}

```

Ejemplo: cree una alarma sencilla con AWS IoT Events

Este modelo de detector es una de las plantillas disponibles en la AWS IoT Events consola. Se incluye aquí para su comodidad.

```

{
    "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"
}

```

Monitorización con alarmas en AWS IoT Events

AWS IoT Events las alarmas le ayudan a supervisar sus datos para detectar cambios. Los datos pueden ser métricas que mide para sus equipos y procesos. Puede crear alarmas que le envíen notificaciones al superarse un umbral. Las alarmas le ayudan a detectar problemas, agilizar el mantenimiento y optimizar el rendimiento de sus equipos y procesos.

Las alarmas son instancias de modelos de alarma. Un modelo de alarma especifica qué detectar, cuándo enviar notificaciones, a quién notificar, etc. También puede especificar una o más [acciones compatibles](#) que se producen cuando cambia el estado de la alarma. AWS IoT Events enruta [los atributos de entrada](#) derivados de sus datos a las alarmas correspondientes. Si los datos que monitorea están fuera del rango especificado, se invoca una alarma. También puede reconocer las alarmas o ponerlas en modo posponer (snooze).

Trabajando con AWS IoT SiteWise

Puede utilizar AWS IoT Events las alarmas para supervisar las propiedades de los activos en AWS IoT SiteWise. AWS IoT SiteWise envía los valores de las propiedades de los activos a AWS IoT Events las alarmas. AWS IoT Events envía el estado de alarma a AWS IoT SiteWise.

AWS IoT SiteWise también admite alarmas externas. Puede elegir alarmas externas si utiliza alarmas externas AWS IoT SiteWise y dispone de una solución que devuelva los datos del estado de la alarma. La alarma externa contiene una propiedad de medición que ingiere los datos del estado de alarma.

AWS IoT SiteWise no evalúa el estado de las alarmas externas. Además, no puede reconocer o posponer una alarma externa si su estado cambia.

Puede utilizar la función SiteWise Monitor para ver el estado de las alarmas externas en los portales SiteWise Monitor.

Para obtener más información, consulte [Supervisión de datos con alarmas](#) en la Guía del AWS IoT SiteWise usuario y [Supervisión con alarmas](#) en la Guía de aplicaciones de SiteWise monitorización.

Flujo de reconocimiento

Al crear un modelo de alarma, puede elegir si desea habilitar el flujo de reconocimiento. Al habilitar el flujo de reconocimiento, su equipo recibirá notificaciones al cambiar el estado de una alarma. Su

equipo puede reconocer la alarma y dejar una nota. Por ejemplo, puede incluir la información de la alarma y las medidas que va a tomar para solucionar el problema. Si los datos que monitorea están fuera del rango especificado, se invoca una alarma.

Las alarmas tienen los siguientes estados:

DISABLED

Cuando la alarma está en el estado DISABLED, no está preparada para evaluar datos. Para activar la alarma, debe cambiarla al estado NORMAL.

NORMAL

Cuando la alarma está en el estado NORMAL, está lista para evaluar datos.

ACTIVE

Si la alarma está en el estado ACTIVE, se invoca la alarma. Los datos que está monitoreando están fuera del rango especificado.

ACKNOWLEDGED

Si la alarma está en el estado ACKNOWLEDGED, se invocó la alarma y usted la reconoció.

LATCHED

Se invocó la alarma, pero usted no la reconoció después de un cierto periodo. La alarma cambia automáticamente al estado NORMAL.

SNOOZE_DISABLED

Cuando la alarma está en el estado SNOOZE_DISABLED, esta se desactiva durante un periodo especificado. Una vez transcurrido el tiempo de aplazamiento, la alarma cambia automáticamente al estado NORMAL.

Creación de un modelo de alarma en AWS IoT Events

Puede utilizar AWS IoT Events las alarmas para supervisar sus datos y recibir notificaciones cuando se supere un umbral. Las alarmas proporcionan parámetros que puede utilizar para crear o configurar un modelo de alarma. Puede usar la AWS IoT Events consola o la AWS IoT Events API para crear o configurar el modelo de alarma. Al configurar el modelo de alarma, los cambios tienen efecto a medida que llegan nuevos datos.

Requisitos

Al crear un modelo de alarma se imponen los siguientes requisitos.

- Puede crear un modelo de alarma para supervisar un atributo de entrada AWS IoT Events o una propiedad de un activo AWS IoT SiteWise.
 - Si decide monitorizar un atributo de entrada en AWS IoT Events, [Cree una entrada para los modelos en AWS IoT Events](#) antes de crear el modelo de alarma.
 - Si decide supervisar una propiedad de activo, debe [crear un modelo de activo](#) AWS IoT SiteWise antes de crear el modelo de alarma.
- Debe tener una función de IAM que permita a la alarma realizar acciones y acceder a AWS los recursos. Para obtener más información, consulte [Configuración de permisos para AWS IoT Events](#).
- Todos los AWS recursos que utilice este tutorial deben estar en la misma AWS región.

Creación de un modelo de alarma (consola)

A continuación, se muestra cómo crear un modelo de alarma para supervisar un AWS IoT Events atributo en la AWS IoT Events consola.

1. Inicie sesión en la [consola de AWS IoT Events](#).
2. En el panel de navegación, seleccione Modelos de alarmas.
3. En la página Modelos de alarmas, seleccione Crear modelo de alarma.
4. En la sección Detalles del modelo de alarma realice lo siguiente:
 - a. Introduzca un nombre único.
 - b. (Opcional) Introduzca una descripción.
5. En la sección Objetivo de la alarma realice lo siguiente:

Important

Para elegir Propiedad de activo de AWS IoT SiteWise , antes debe haber creado un modelo de activo en AWS IoT SiteWise.

- a. Seleccione Atributo de entrada de AWS IoT Events .

- b. Seleccione la entrada.
- c. Seleccione la clave del atributo de entrada. Este atributo de entrada se utiliza como clave para crear la alarma. AWS IoT Events enruta las entradas asociadas a esta clave a la alarma.

 Important

Si la carga del mensaje de entrada no contiene esta clave de atributo de entrada o si la clave no se encuentra en la misma ruta JSON especificada en la clave, la ingesta del mensaje en AWS IoT Events fallará.

6. En la sección de definiciones de umbrales, se definen el atributo de entrada, el valor umbral y el operador de comparación que se AWS IoT Events utilizan para cambiar el estado de la alarma.

- a. En Atributo de entrada, elija el atributo que desee monitorear.

Toda vez que este atributo de entrada reciba nuevos datos, estos se evaluarán para determinar el estado de la alarma.

- b. En Operador, elija el operador de comparación. El operador compara su atributo de entrada con el valor umbral de del mismo.

Puede elegir entre estas opciones:

- > mayor que
 - >= mayor o igual que
 - < menor que
 - <= menor o igual a
 - = igual a
 - != distinto de
- c. Para el valor umbral, introduzca un número o elija un atributo en AWS IoT Events las entradas. AWS IoT Events compara este valor con el valor del atributo de entrada que elija.
 - d. (Opcional) Para Gravedad, utilice un número que su equipo comprenda y refleje la gravedad de la alarma.
7. (Opcional) En la sección Configuración de notificaciones, configure los parámetros de notificación de la alarma.

Puede añadir hasta 10 notificaciones. En Notificación 1, haga lo siguiente:

- a. En Protocolo, elija entre las siguientes opciones:
 - Correo electrónico y texto: La alarma envía una notificación por SMS y una por correo electrónico.
 - Correo electrónico: La alarma envía una notificación por correo electrónico.
 - Texto: La alarma envía una notificación por SMS.
- b. En Remitente, especifique la dirección de correo electrónico que puede enviar notificaciones sobre esta alarma.

Para añadir más direcciones de correo electrónico a la lista de remitentes, seleccione Añadir remitente.

- c. (Opcional) En Destinatario, elija el destinatario.

Para añadir más usuarios a su lista de destinatarios, elija Añadir nuevo usuario. Debe añadir los nuevos usuarios a su almacén del Centro de identidades de IAM antes de poder añadirlos a su modelo de alarma. Para obtener más información, consulte [Gestione el acceso al IAM Identity Center de los destinatarios de las alarmas en AWS IoT Events](#).

- d. (Opcional) En Mensaje personalizado adicional, introduzca un mensaje que describa qué detecta la alarma y qué acciones deben realizar los destinatarios.
8. En la sección Instancia, puede activar o desactivar todas las instancias de alarma que se creen basándose en este modelo de alarma.
 9. En la sección Ajustes avanzados, haga lo siguiente:

- a. En Flujo de reconocimiento, puede habilitar o deshabilitar las notificaciones.
 - Si elige Habilitado, recibirá una notificación cuando cambie el estado de la alarma. Debe reconocer la notificación para que el estado de alarma vuelva a normal.
 - Si elige Deshabilitado, no se requiere ninguna acción. La alarma cambia automáticamente al estado normal cuando la medición vuelve al rango especificado.

Para obtener más información, consulte [Flujo de reconocimiento](#).

- b. En Permisos, elija una de las siguientes opciones:

- Puede crear un nuevo rol a partir de plantillas de AWS políticas y crear AWS IoT Events automáticamente un rol de IAM para usted.
- Puede utilizar un rol de IAM existente que permita a este modelo de alarma realizar acciones y acceder a otros AWS recursos.

Para obtener más información, consulte [Administración de identidad y acceso para AWS IoT Events](#).

- c. Para configurar las notificaciones adicionales, puede editar su AWS Lambda función para gestionar las notificaciones de alarma. Elige una de las siguientes opciones para tu AWS Lambda función:
 - Crear una AWS Lambda función nueva: AWS IoT Events crea una AWS Lambda función nueva para usted.
 - Utilizar una AWS Lambda función existente: utilice una AWS Lambda función existente eligiendo un nombre de AWS Lambda función.

Para obtener más información sobre las acciones posibles, consulte [AWS IoT Events trabajando con otros servicios AWS](#).

- d. (Opcional) En la acción Establecer estado, puede añadir una o más AWS IoT Events acciones para llevarlas a cabo cuando cambie el estado de la alarma.
10. (Opcional) Puede añadir Etiquetas para gestionar sus alarmas. Para obtener más información, consulte [Etiquetado de los recursos de AWS IoT Events](#).
 11. Seleccione Crear.

Responder a las alarmas en AWS IoT Events

Responder a las alarmas de forma eficaz es un aspecto importante de la gestión de los sistemas de IoT con AWS IoT Events. Explore varias formas de configurar y gestionar las alarmas, entre las que se incluyen la configuración de los canales de notificación, la definición de los procedimientos de escalamiento y la implementación de acciones de respuesta automatizadas. Aprenda a crear condiciones de alarma matizadas, a priorizar las alertas y a integrarlas con otros AWS servicios para crear un sistema de gestión de alarmas con capacidad de respuesta para sus aplicaciones de IoT.

Si activó [Flujo de reconocimiento](#), recibirá notificaciones al cambiar el estado de la alarma. Para responder a la alarma, puede reconocerla, deshabilitarla, habilitarla, restablecerla o posponerla.

Console

A continuación, se muestra cómo responder a una alarma en la consola de AWS IoT Events .

1. Inicie sesión en la [consola de AWS IoT Events](#).
2. En el panel de navegación, seleccione Modelos de alarmas.
3. Seleccione el modelo de alarma deseado.
4. En la sección Lista de alarmas, elija la alarma deseada.
5. En Acciones, puede elegir una de las siguientes opciones:
 - Reconocer: La alarma cambia al estado ACKNOWLEDGED.
 - Deshabilitar: La alarma pasa al estado DISABLED.
 - Habilitar: La alarma cambia al estado NORMAL.
 - Reiniciar: La alarma cambia al estado NORMAL.
 - Posponer, y luego haga lo siguiente:
 1. Elija la Duración de snooze o introduzca una Duración de Snooze personalizada.
 2. Seleccione Guardar.

La alarma cambia al estado SNOOZE_DISABLED

Para obtener más información sobre los estados de alarma, consulte [Flujo de reconocimiento](#).

API

Para responder a una o más alarmas, puede utilizar las siguientes operaciones de AWS IoT Events API:

- [BatchAcknowledgeAlarm](#)
- [BatchDisableAlarm](#)
- [BatchEnableAlarm](#)
- [BatchResetAlarm](#)
- [BatchSnoozeAlarm](#)

Administrar las notificaciones de alarmas en AWS IoT Events

AWS IoT Events se integra con Lambda y ofrece capacidades de procesamiento de eventos personalizadas. En esta sección, se explica cómo utilizar las funciones Lambda en sus modelos de AWS IoT Events detector, lo que le permite ejecutar una lógica compleja, interactuar con servicios externos e implementar una gestión de eventos sofisticada.

AWS IoT Events utiliza una función Lambda para gestionar las notificaciones de alarma. Puede utilizar la función Lambda proporcionada por AWS IoT Events o crear una nueva.

Temas

- [Creación de una función Lambda en AWS IoT Events](#)
- [Uso de la función de Lambda proporcionada por AWS IoT Events](#)
- [Gestione el acceso al IAM Identity Center de los destinatarios de las alarmas en AWS IoT Events](#)

Creación de una función Lambda en AWS IoT Events

AWS IoT Events proporciona una función Lambda que permite a las alarmas enviar y recibir notificaciones por correo electrónico y SMS.

Requisitos

Se imponen los siguientes requisitos al crear una función de Lambda para las alarmas:

- Si la alarma envía notificaciones por SMS, asegúrese de que Amazon SNS esté configurado para entregar mensajes SMS.
 - Para obtener más información, consulte la siguiente documentación sobre :
 - [Mensajes de texto móviles con Amazon SNS](#) e [identidades de origen para los mensajes SMS de Amazon SNS en la guía para desarrolladores de Amazon Simple Notification Service](#).
 - [¿Qué es la mensajería SMS para usuarios finales de AWS?](#) en la Guía AWS SMS del usuario.
- Si la alarma envía notificaciones por correo electrónico o SMS, debe tener una función de IAM que le permita AWS Lambda trabajar con Amazon SES y Amazon SNS.

Ejemplo de política:

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "ses:GetIdentityVerificationAttributes",
        "ses:SendEmail",
        "ses:VerifyEmailIdentity"
      ],
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "sns:Publish",
        "sns:OptInPhoneNumber",
        "sns:CheckIfPhoneNumberIsOptedOut",
        "sms-voice:DescribeOptedOutNumbers"
      ],
      "Resource": "*"
    },
    {
      "Effect": "Deny",
      "Action": "sns:Publish",
      "Resource": "arn:aws:sns:*:*:*"
    },
    {
      "Effect" : "Allow",
      "Action" : [
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:PutLogEvents"
      ],
      "Resource" : "*"
    }
  ]
}

```

- Debe elegir la misma AWS región para ambos AWS IoT Events tipos y. AWS Lambda Para obtener la lista de regiones admitidas, consulte [Puntos de conexión y cuotas de AWS IoT Events](#) y [Puntos de conexión y cuotas de AWS Lambda](#) en Referencia general de Amazon Web Services.

Implementar una función Lambda para usar AWS IoT EventsAWS CloudFormation

En este tutorial se utiliza una AWS CloudFormation plantilla para implementar una función Lambda. Esta plantilla crea automáticamente un rol de IAM que permite a la función de Lambda trabajar con Amazon SES y Amazon SNS.

A continuación, se muestra cómo usar el AWS Command Line Interface (AWS CLI) para crear una CloudFormation pila.

1. En la terminal de tu dispositivo, ejecuta `aws --version` para comprobar si has instalado el AWS CLI. Para obtener más información, consulte [Instalación o actualización de la versión más reciente de la AWS CLI](#) en la Guía del usuario de AWS Command Line Interface .
2. Ejecute `aws configure list` para comprobar si ha configurado AWS CLI la AWS región que tiene todos sus AWS recursos para este tutorial. Para obtener más información, consulte [Establecer y ver los ajustes de configuración mediante comandos](#) en la Guía del AWS Command Line Interface usuario
3. Descargue la CloudFormation plantilla [NotificationLambda.Template.yaml.zip](#).

Note

Si encuentra dificultades para descargar el archivo, la plantilla también está disponible en [CloudFormation plantilla](#).

4. Descomprima el contenido y guárdelo localmente como `notificationLambda.template.yaml`.
5. Abra un terminal en su dispositivo y vaya hasta el directorio en que descargó el archivo `notificationLambda.template.yaml`.
6. Para crear una pila CloudFormation , ejecuta el siguiente comando:

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

Puede modificar esta CloudFormation plantilla para personalizar la función Lambda y su comportamiento.

Note

AWS Lambda Reintenta los errores de función dos veces. Si la función no tiene capacidad suficiente para gestionar todas las solicitudes entrantes, los eventos podrían esperar en la cola durante horas o días para su envío a la función. Puede configurar una cola de mensajes no entregados (DLQ) en la función para capturar los eventos que no se procesaron correctamente. Para obtener más información, consulte [Invocación asíncrona](#) en la Guía para desarrolladores de AWS Lambda .

También puede crear o configurar la pila en la CloudFormation consola. Para obtener más información, consulte [Trabajo con pilas](#), en la Guía del usuario de AWS CloudFormation .

Creación de una función Lambda personalizada para AWS IoT Events

Puede crear una función de Lambda o modificar la proporcionada con AWS IoT Events.

Se imponen los siguientes requisitos al crear una función de Lambda personalizada.

- Agregue permisos que permitan a la función Lambda realizar acciones específicas y acceder AWS a los recursos.
- Si usa la función Lambda proporcionada por AWS IoT Events, asegúrese de elegir el motor de ejecución de Python 3.7.

Ejemplo de función de Lambda:

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

```

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

# Check whether the phone holder has opted out of receiving SMS messages from your
account
def check_phone_number(phone_number):
    try:
        result = sns.check_if_phone_number_is_opted_out(phoneNumber=phone_number)
        if (result['isOptedOut']):
            logger.info('phoneNumber {} is not opt in of receiving SMS messages. Phone
number must be opt in first.'.format(phone_number))
            return False
        return True
    except Exception as e:
        logging.error('Your phone number {} must be in E.164 format in SS0. Exception
thrown: {}'.format(phone_number, e))
        return False

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

def lambda_handler(event, context):
    logging.info('Received event: ' + json.dumps(event))
    nep = json.loads(event.get('notificationEventPayload'))
    alarm_state = nep['alarmState']
    default_msg = 'Alarm ' + alarm_state['stateName'] + '\n'
    timestamp =
datetime.datetime.utcnow().timestamp(float(nep['stateUpdateTime'])/1000).strftime('%Y-
%m-%d %H:%M:%S')

```

```

alarm_msg = "{} {} {} at {} UTC ".format(nep['alarmModelName'], nep.get('keyValue',
'Singleton'), alarm_state['stateName'], timestamp)
default_msg += 'Sev: ' + str(nep['severity']) + '\n'
if (alarm_state['ruleEvaluation']):
    property = alarm_state['ruleEvaluation']['simpleRule']['inputProperty']
    default_msg += 'Current Value: ' + str(property) + '\n'
    operator = alarm_state['ruleEvaluation']['simpleRule']['operator']
    threshold = alarm_state['ruleEvaluation']['simpleRule']['threshold']
    alarm_msg += '({} {} {}>'.format(str(property), operator, str(threshold))
default_msg += alarm_msg + '\n'

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

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

```

```
else:
    sns.publish(PhoneNumber=phone_number, Message=sns_msg)
logger.info('SNS messages have been sent')
```

Para obtener más información, consulte [¿Qué es AWS Lambda?](#) en la Guía para desarrolladores de AWS Lambda .

CloudFormation plantilla

Utilice la siguiente CloudFormation plantilla para crear la función Lambda.

```
AWSTemplateFormatVersion: '2010-09-09'
Description: 'Notification Lambda for Alarm Model'
Resources:
  NotificationLambdaRole:
    Type: AWS::IAM::Role
    Properties:
      AssumeRolePolicyDocument:
        Statement:
          - Effect: Allow
            Principal:
              Service: lambda.amazonaws.com
            Action: sts:AssumeRole
      Path: "/"
      ManagedPolicyArns:
        - 'arn:aws:iam::aws:policy/AWSLambdaExecute'
    Policies:
      - PolicyName: "NotificationLambda"
        PolicyDocument:
          Version: "2012-10-17"
          Statement:
            - Effect: "Allow"
              Action:
                - "ses:GetIdentityVerificationAttributes"
                - "ses:SendEmail"
                - "ses:VerifyEmailIdentity"
              Resource: "*"
            - Effect: "Allow"
              Action:
                - "sns:Publish"
                - "sns:OptInPhoneNumber"
                - "sns:CheckIfPhoneNumberIsOptedOut"
                - "sms-voice:DescribeOptedOutNumbers"
```

```

        Resource: "*"
    - Effect: "Deny"
      Action:
        - "sns:Publish"
      Resource: "arn:aws:sns:*:*:*"
NotificationLambdaFunction:
  Type: AWS::Lambda::Function
  Properties:
    Role: !GetAtt NotificationLambdaRole.Arn
    Runtime: python3.7
    Handler: index.lambda_handler
    Timeout: 300
    MemorySize: 3008
    Code:
      ZipFile: |
        import boto3
        import json
        import logging
        import datetime
        logger = logging.getLogger()
        logger.setLevel(logging.INFO)
        ses = boto3.client('ses')
        sns = boto3.client('sns')
        def check_value(target):
            if target:
                return True
            return False

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

        # Check whether the phone holder has opted out of receiving SMS messages from
        your account

```

```

def check_phone_number(phone_number):
    try:
        result = sns.check_if_phone_number_is_opted_out(phoneNumber=phone_number)
        if (result['isOptedOut']):
            logger.info('phoneNumber {} is not opt in of receiving SMS messages.
Phone number must be opt in first.'.format(phone_number))
            return False
        return True
    except Exception as e:
        logging.error('Your phone number {} must be in E.164 format in SS0.
Exception thrown: {}'.format(phone_number, e))
        return False

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

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

    emails = event.get('emailConfigurations', [])
    logger.info('Start Sending Emails')
    for email in emails:
        from_addr = email.get('from')
        to_addrs = 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')

```

Uso de la función de Lambda proporcionada por AWS IoT Events

Con las notificaciones de alarma, puede utilizar la función Lambda proporcionada por AWS IoT Events para gestionar las notificaciones de alarma.

Se imponen los siguientes requisitos al utilizar la función de Lambda proporcionada por AWS IoT Events para administrar sus notificaciones de alarma:

- Debe verificar la dirección de correo electrónico que envía las notificaciones por correo electrónico en Amazon Simple Email Service (Amazon SES). Para obtener más información, consulte

[Verificación de una identidad de dirección de correo electrónico](#) en la Guía para desarrolladores de Amazon Simple Email Service.

Si recibe un enlace de verificación, pulse el enlace para verificar su dirección de correo electrónico. También debería controlar su carpeta de correo no deseado en busca del correo electrónico de verificación.

- Si su alarma envía notificaciones por SMS, para los números de teléfono debe utilizar el formato de número de teléfono internacional E.164. Este formato contiene `+<country-calling-code><area-code><phone-number>`.

Ejemplos de números de teléfono:

País	Número de teléfono local	Número en formato E.164
Estados Unidos	206-555-0100	+12065550100
Reino Unido	020-1234-1234	+442012341234
Lituania	8+601+12345	+37060112345

Para encontrar el prefijo telefónico de un país, visite countrycode.org.

La función Lambda proporcionada por AWS IoT Events comprueba si utiliza números de teléfono con formato E.164. Sin embargo, no verifica los números de teléfono. Si está seguro de haber introducido números de teléfono correctos pero no ha recibido notificaciones por SMS, debe ponerse en contacto con las compañías telefónicas. Es posible que los operadores bloqueen los mensajes.

Gestione el acceso al IAM Identity Center de los destinatarios de las alarmas en AWS IoT Events

AWS IoT Events AWS IAM Identity Center se utiliza para gestionar el acceso SSO de los destinatarios de las alarmas. La implementación del IAM Identity Center para los destinatarios de las AWS IoT Events notificaciones puede mejorar la seguridad y la experiencia del usuario. Para que la alarma pueda enviar notificaciones a los destinatarios, debe habilitar el Centro de identidades de IAM y añadir destinatarios a su almacén del Centro de identidades de IAM. Para obtener más información, consulte [Adición de usuarios](#) en la Guía de usuarios de AWS IAM Identity Center .

 Important

- Debe elegir la misma AWS región para AWS IoT Events el centro de identidad de IAM y para él. AWS Lambda
- AWS Organizations solo admite una región de IAM Identity Center a la vez. Si desea que el Centro de identidades de IAM esté disponible en una región diferente, primero debe eliminar la configuración actual del Centro de identidades de IAM. Para obtener más información, consulte [Datos regionales del Centro de identidades de IAM](#) en la Guía del usuario de AWS IAM Identity Center .

Seguridad en AWS IoT Events

La seguridad en la nube AWS es la máxima prioridad. Como AWS cliente, usted se beneficia de una arquitectura de centro de datos y red diseñada para cumplir con los requisitos de las organizaciones más sensibles a la seguridad.

La seguridad es una responsabilidad compartida entre usted AWS y usted. El [modelo de responsabilidad compartida](#) la describe como seguridad de la nube y seguridad en la nube:

- Seguridad de la nube: AWS es responsable de proteger la infraestructura que ejecuta AWS los servicios en la AWS nube. AWS también le proporciona servicios que puede utilizar de forma segura. Auditores externos prueban y verifican periódicamente la eficacia de nuestra seguridad en el marco de los [programas de conformidad de AWS](#). Para obtener más información sobre los programas de cumplimiento aplicables AWS IoT Events, consulte los [AWS servicios incluidos en el ámbito de aplicación por programa de cumplimiento](#).
- Seguridad en la nube: su responsabilidad viene determinada por el AWS servicio que utilice. Usted también es responsable de otros factores incluida la confidencialidad de los datos, los requisitos de la empresa y la legislación y los reglamentos aplicables.

Esta documentación le ayudará a entender cómo aplicar el modelo de responsabilidad compartida cuando lo utilice AWS IoT Events. Los siguientes temas muestran cómo configurarlo AWS IoT Events para cumplir sus objetivos de seguridad y conformidad. También aprenderá a utilizar otros AWS servicios que pueden ayudarle a supervisar y proteger sus AWS IoT Events recursos.

Temas

- [Gestión de identidad y acceso para AWS IoT Events](#)
- [Supervisión AWS IoT Events para mantener la confiabilidad, la disponibilidad y el rendimiento](#)
- [Validación de conformidad para AWS IoT Events](#)
- [Resiliencia en AWS IoT Events](#)
- [Seguridad de la infraestructura en AWS IoT Events](#)

Gestión de identidad y acceso para AWS IoT Events

AWS Identity and Access Management (IAM) es un AWS servicio que ayuda al administrador a controlar de forma segura el acceso a AWS los recursos. Los administradores de IAM controlan

quién puede autenticarse (iniciar sesión) y quién puede autorizarse (tener permisos) para usar los recursos. AWS IoT Events La IAM es un AWS servicio que puede utilizar sin coste adicional.

Temas

- [Público](#)
- [Autenticación con identidades](#)
- [Administración de acceso mediante políticas](#)
- [Más información sobre la gestión de identidades y accesos](#)
- [¿Cómo AWS IoT Events funciona con IAM](#)
- [AWS IoT Events ejemplos de políticas basadas en la identidad](#)
- [Prevención policial confusa entre servicios para AWS IoT Events](#)
- [Solucionar problemas de AWS IoT Events identidad y acceso](#)

Público

La forma de usar AWS Identity and Access Management (IAM) varía según el trabajo en el que se desempeñe. AWS IoT Events

Usuario del servicio: si utiliza el AWS IoT Events servicio para realizar su trabajo, el administrador le proporcionará las credenciales y los permisos que necesita. A medida que vaya utilizando más AWS IoT Events funciones para realizar su trabajo, es posible que necesite permisos adicionales. Entender cómo se administra el acceso puede ayudarle a solicitar los permisos correctos al administrador. Si no puede acceder a una característica en AWS IoT Events, consulte [Solucionar problemas de AWS IoT Events identidad y acceso](#).

Administrador de servicios: si estás a cargo de AWS IoT Events los recursos de tu empresa, probablemente tengas acceso total a ellos AWS IoT Events. Su trabajo consiste en determinar a qué AWS IoT Events funciones y recursos deben acceder los usuarios del servicio. Luego, debe enviar solicitudes a su gestor de IAM para cambiar los permisos de los usuarios de su servicio. Revise la información de esta página para conocer los conceptos básicos de IAM. Para obtener más información sobre cómo su empresa puede utilizar la IAM AWS IoT Events, consulte [¿Cómo AWS IoT Events funciona con IAM](#).

Administrador de IAM: si es un administrador de IAM, es posible que quiera conocer más detalles sobre cómo escribir políticas para administrar el acceso a AWS IoT Events. Para ver ejemplos de

políticas AWS IoT Events basadas en la identidad que puede utilizar en IAM, consulte. [AWS IoT Events ejemplos de políticas basadas en la identidad](#)

Autenticación con identidades

La autenticación es la forma de iniciar sesión AWS con sus credenciales de identidad. Debe estar autenticado (con quien haya iniciado sesión AWS) como usuario de IAM o asumiendo una función de IAM. Usuario raíz de la cuenta de AWS

Puede iniciar sesión AWS como una identidad federada mediante las credenciales proporcionadas a través de una fuente de identidad. AWS IAM Identity Center Los usuarios (Centro de identidades de IAM), la autenticación de inicio de sesión único de su empresa y sus credenciales de Google o Facebook son ejemplos de identidades federadas. Al iniciar sesión como una identidad federada, su gestor habrá configurado previamente la federación de identidades mediante roles de IAM. Cuando accedes AWS mediante la federación, estás asumiendo un rol de forma indirecta.

Según el tipo de usuario que sea, puede iniciar sesión en el portal AWS Management Console o en el de AWS acceso. Para obtener más información sobre cómo iniciar sesión AWS, consulte [Cómo iniciar sesión Cuenta de AWS en su](#) Guía del AWS Sign-In usuario.

Si accede AWS mediante programación, AWS proporciona un kit de desarrollo de software (SDK) y una interfaz de línea de comandos (CLI) para firmar criptográficamente sus solicitudes con sus credenciales. Si no utilizas AWS herramientas, debes firmar las solicitudes tú mismo. Para obtener más información sobre la firma de solicitudes, consulte [AWS Signature Versión 4 para solicitudes API](#) en la Guía del usuario de IAM.

Independientemente del método de autenticación que use, es posible que deba proporcionar información de seguridad adicional. Por ejemplo, le AWS recomienda que utilice la autenticación multifactor (MFA) para aumentar la seguridad de su cuenta. Para obtener más información, consulte [Autenticación multifactor](#) en la Guía del usuario de AWS IAM Identity Center y [Autenticación multifactor AWS en IAM](#) en la Guía del usuario de IAM.

Cuenta de AWS usuario root

Al crear una Cuenta de AWS, comienza con una identidad de inicio de sesión que tiene acceso completo a todos Servicios de AWS los recursos de la cuenta. Esta identidad se denomina usuario Cuenta de AWS raíz y se accede a ella iniciando sesión con la dirección de correo electrónico y la contraseña que utilizaste para crear la cuenta. Recomendamos encarecidamente que no utiliza el usuario raíz para sus tareas diarias. Proteja las credenciales del usuario raíz y utilícelas solo para las

tareas que solo el usuario raíz pueda realizar. Para ver la lista completa de las tareas que requieren que inicie sesión como usuario raíz, consulte [Tareas que requieren credenciales de usuario raíz](#) en la Guía del usuario de IAM.

Usuarios y grupos de IAM

Un [usuario de IAM](#) es una identidad propia Cuenta de AWS que tiene permisos específicos para una sola persona o aplicación. Siempre que sea posible, recomendamos emplear credenciales temporales, en lugar de crear usuarios de IAM que tengan credenciales de larga duración como contraseñas y claves de acceso. No obstante, si tiene casos de uso específicos que requieran credenciales de larga duración con usuarios de IAM, recomendamos rotar las claves de acceso. Para más información, consulta [Rotar las claves de acceso periódicamente para casos de uso que requieran credenciales de larga duración](#) en la Guía del usuario de IAM.

Un [grupo de IAM](#) es una identidad que especifica un conjunto de usuarios de IAM. No puedes iniciar sesión como grupo. Puedes usar los grupos para especificar permisos para varios usuarios a la vez. Los grupos facilitan la administración de los permisos para grandes conjuntos de usuarios. Por ejemplo, puede asignar un nombre a un grupo IAMAdminsy concederle permisos para administrar los recursos de IAM.

Los usuarios son diferentes de los roles. Un usuario se asocia exclusivamente a una persona o aplicación, pero la intención es que cualquier usuario pueda asumir un rol que necesite. Los usuarios tienen credenciales de larga duración permanentes; no obstante, los roles proporcionan credenciales temporales. Para obtener más información, consulte [Casos de uso para usuarios de IAM](#) en la Guía del usuario de IAM.

Roles de IAM

Un [rol de IAM](#) es una identidad dentro de usted Cuenta de AWS que tiene permisos específicos. Es similar a un usuario de IAM, pero no está asociado a una persona determinada. Para asumir temporalmente un rol de IAM en el AWS Management Console, puede [cambiar de un rol de usuario a uno de IAM](#) (consola). Puedes asumir un rol llamando a una operación de AWS API AWS CLI o usando una URL personalizada. Para más información sobre los métodos para el uso de roles, consulta [Métodos para asumir un rol](#) en la Guía del usuario de IAM.

Los roles de IAM con credenciales temporales son útiles en las siguientes situaciones:

- **Acceso de usuario federado:** para asignar permisos a una identidad federada, puedes crear un rol y definir sus permisos. Cuando se autentica una identidad federada, se asocia la identidad al rol y se le conceden los permisos define el rol. Para obtener información acerca de roles de

federación, consulte [Crear un rol para un proveedor de identidad de terceros \(federación\)](#) en la Guía de usuario de IAM. Si utiliza el IAM Identity Center, debe configurar un conjunto de permisos. IAM Identity Center correlaciona el conjunto de permisos con un rol en IAM para controlar a qué puedes acceder las identidades después de autenticarse. Para obtener información acerca de los conjuntos de permisos, consulta [Conjuntos de permisos](#) en la Guía del usuario de AWS IAM Identity Center .

- **Permisos de usuario de IAM temporales:** un usuario de IAM puedes asumir un rol de IAM para recibir temporalmente permisos distintos que le permitan realizar una tarea concreta.
- **Acceso entre cuentas:** puedes utilizar un rol de IAM para permitir que alguien (una entidad principal de confianza) de otra cuenta acceda a los recursos de la cuenta. Los roles son la forma principal de conceder acceso entre cuentas. Sin embargo, con algunas Servicios de AWS, puedes adjuntar una política directamente a un recurso (en lugar de usar un rol como proxy). Para obtener información acerca de la diferencia entre los roles y las políticas basadas en recursos para el acceso entre cuentas, consulta [Acceso a recursos entre cuentas en IAM](#) en la Guía del usuario de IAM.
- **Acceso entre servicios:** algunos Servicios de AWS utilizan funciones en otros Servicios de AWS. Por ejemplo, cuando realizas una llamada en un servicio, es habitual que ese servicio ejecute aplicaciones en Amazon EC2 o almacene objetos en Amazon S3. Es posible que un servicio haga esto usando los permisos de la entidad principal, usando un rol de servicio o usando un rol vinculado al servicio.
- **Sesiones de acceso directo (FAS):** cuando utilizas un usuario o un rol de IAM para realizar acciones en AWS ellas, se te considera principal. Cuando utiliza algunos servicios, es posible que realice una acción que desencadene otra acción en un servicio diferente. El FAS utiliza los permisos del principal que llama Servicio de AWS y los solicita Servicio de AWS para realizar solicitudes a los servicios descendentes. Las solicitudes de FAS solo se realizan cuando un servicio recibe una solicitud que requiere interacciones con otros Servicios de AWS recursos para completarse. En este caso, debe tener permisos para realizar ambas acciones. Para obtener información sobre las políticas a la hora de realizar solicitudes de FAS, consulta [Reenviar sesiones de acceso](#).
- **Rol de servicio:** un rol de servicio es un [rol de IAM](#) que adopta un servicio para realizar acciones en su nombre. Un administrador de IAM puede crear, modificar y eliminar un rol de servicio desde IAM. Para obtener más información, consulte [Creación de un rol para delegar permisos a un Servicio de AWS](#) en la Guía del usuario de IAM.
- **Función vinculada al servicio:** una función vinculada a un servicio es un tipo de función de servicio que está vinculada a un. Servicio de AWS El servicio puedes asumir el rol para realizar

una acción en su nombre. Los roles vinculados al servicio aparecen en usted Cuenta de AWS y son propiedad del servicio. Un administrador de IAM puedes ver, pero no editar, los permisos de los roles vinculados a servicios.

- Aplicaciones que se ejecutan en Amazon EC2: puedes usar un rol de IAM para administrar las credenciales temporales de las aplicaciones que se ejecutan en una EC2 instancia y realizan AWS CLI solicitudes a la AWS API. Esto es preferible a almacenar las claves de acceso en la EC2 instancia. Para asignar un AWS rol a una EC2 instancia y ponerlo a disposición de todas sus aplicaciones, debe crear un perfil de instancia adjunto a la instancia. Un perfil de instancia contiene el rol y permite que los programas que se ejecutan en la EC2 instancia obtengan credenciales temporales. Para obtener más información, consulte [Usar un rol de IAM para conceder permisos a las aplicaciones que se ejecutan en EC2 instancias de Amazon](#) en la Guía del usuario de IAM.

Administración de acceso mediante políticas

El acceso se controla AWS creando políticas y adjuntándolas a AWS identidades o recursos. Una política es un objeto AWS que, cuando se asocia a una identidad o un recurso, define sus permisos. AWS evalúa estas políticas cuando un director (usuario, usuario raíz o sesión de rol) realiza una solicitud. Los permisos en las políticas determinan si la solicitud se permite o se deniega. La mayoría de las políticas se almacenan AWS como documentos JSON. Para obtener más información sobre la estructura y el contenido de los documentos de política JSON, consulta [Información general de políticas JSON](#) en la Guía del usuario de IAM.

Los administradores pueden usar las políticas de AWS JSON para especificar quién tiene acceso a qué. Es decir, qué entidad principal puede realizar acciones en qué recursos y en qué condiciones.

De forma predeterminada, los usuarios y los roles no tienen permisos. Un administrador de IAM puedes crear políticas de IAM para conceder permisos a los usuarios para realizar acciones en los recursos que necesitan. A continuación, el administrador puedes añadir las políticas de IAM a roles y los usuarios puedes asumirlos.

Las políticas de IAM definen permisos para una acción independientemente del método que se utiliza para realizar la operación. Por ejemplo, suponga que dispone de una política que permite la acción `iam:GetRole`. Un usuario con esa política puede obtener información sobre el rol de la API AWS Management Console AWS CLI, la o la AWS API.

Políticas basadas en identidades

Las políticas basadas en identidad son documentos de políticas de permisos JSON que puedes asociar a una identidad, como un usuario de IAM, un grupo de usuarios o un rol. Estas políticas controlan qué acciones pueden realizar los usuarios y los roles, en qué recursos y en qué condiciones. Para obtener más información sobre cómo crear una política basada en identidad, consulte [Creación de políticas de IAM](#) en la Guía del usuario de IAM.

Las políticas basadas en identidades puedes clasificarse además como políticas insertadas o políticas administradas. Las políticas insertadas se integran directamente en un único usuario, grupo o rol. Las políticas administradas son políticas independientes que puede adjuntar a varios usuarios, grupos y roles de su Cuenta de AWS empresa. Las políticas administradas incluyen políticas AWS administradas y políticas administradas por el cliente. Para obtener más información sobre cómo elegir una política administrada o una política insertada, consulte [Elegir entre políticas administradas y políticas insertadas](#) en la Guía del usuario de IAM.

Otros tipos de políticas

AWS admite tipos de políticas adicionales y menos comunes. Estos tipos de políticas puedes establecer el máximo de permisos que los tipos de políticas más frecuentes le conceden.

- **Límites de permisos:** un límite de permisos es una característica avanzada que le permite establecer los permisos máximos que una política basada en identidad puedes conceder a una entidad de IAM (usuario o rol de IAM). Puedes establecer un límite de permisos para una entidad. Los permisos resultantes son la intersección de las políticas basadas en la identidad de la entidad y los límites de permisos. Las políticas basadas en recursos que especifiquen el usuario o rol en el campo `Principal` no estarán restringidas por el límite de permisos. Una denegación explícita en cualquiera de estas políticas anulará el permiso. Para obtener más información sobre los límites de los permisos, consulta [Límites de permisos para las entidades de IAM](#) en la Guía del usuario de IAM.
- **Políticas de control de servicios (SCPs):** SCPs son políticas de JSON que especifican los permisos máximos para una organización o unidad organizativa (OU). AWS Organizations es un servicio para agrupar y administrar de forma centralizada varios de los Cuentas de AWS que son propiedad de su empresa. Si habilitas todas las funciones de una organización, puedes aplicar políticas de control de servicios (SCPs) a una o a todas tus cuentas. El SCP limita los permisos de las entidades en las cuentas de los miembros, incluidas las de cada una Usuario raíz de la cuenta de AWS. Para obtener más información sobre Organizations SCPs, consulte las [políticas de control de servicios](#) en la Guía del AWS Organizations usuario.

- **Políticas de control de recursos (RCPs):** RCPs son políticas de JSON que puedes usar para establecer los permisos máximos disponibles para los recursos de tus cuentas sin actualizar las políticas de IAM asociadas a cada recurso que poseas. El RCP limita los permisos de los recursos en las cuentas de los miembros y puede afectar a los permisos efectivos de las identidades, incluidos los permisos Usuario raíz de la cuenta de AWS, independientemente de si pertenecen a su organización. Para obtener más información sobre Organizations e RCPs incluir una lista de Servicios de AWS ese apoyo RCPs, consulte [Políticas de control de recursos \(RCPs\)](#) en la Guía del AWS Organizations usuario.
- **Políticas de sesión:** las políticas de sesión son políticas avanzadas que se pasan como parámetro cuando se crea una sesión temporal mediante programación para un rol o un usuario federado. Los permisos de la sesión resultantes son la intersección de las políticas basadas en identidades del rol y las políticas de la sesión. Los permisos también puedes proceder de una política en función de recursos. Una denegación explícita en cualquiera de estas políticas anulará el permiso. Para más información, consulta [Políticas de sesión](#) en la Guía del usuario de IAM.

Varios tipos de políticas

Cuando se aplican varios tipos de políticas a una solicitud, los permisos resultantes son más complicados de entender. Para saber cómo se AWS determina si se debe permitir una solicitud cuando se trata de varios tipos de políticas, consulte la [lógica de evaluación de políticas](#) en la Guía del usuario de IAM.

Más información sobre la gestión de identidades y accesos

Para obtener más información sobre la administración de identidades y accesos AWS IoT Events, consulte las siguientes páginas:

- [¿Cómo AWS IoT Events funciona con IAM](#)
- [Solucionar problemas de AWS IoT Events identidad y acceso](#)

¿Cómo AWS IoT Events funciona con IAM

Antes de utilizar IAM para gestionar el acceso AWS IoT Events, debe comprender las funciones de IAM disponibles para su uso. AWS IoT EventsPara obtener una visión general de cómo funcionan con IAM AWS IoT Events y otros AWS servicios, consulte los [AWS servicios que funcionan con IAM en la Guía del usuario de IAM](#).

Temas

- [AWS IoT Events políticas basadas en la identidad](#)
- [Políticas de AWS IoT Events basadas en recursos](#)
- [Autorización basada en etiquetas de AWS IoT Events](#)
- [AWS IoT Events Funciones de IAM](#)

AWS IoT Events políticas basadas en la identidad

Con las políticas basadas en identidad de IAM, puede especificar las acciones permitidas o denegadas y los recursos, además de las condiciones en las que se permiten o deniegan las acciones. AWS IoT Events admite acciones, recursos y claves de condiciones específicos. Para obtener más información acerca de los elementos que utiliza en una política de JSON, consulte [Referencia de los elementos de las políticas de JSON de IAM](#) en la Guía del usuario de IAM.

Acciones

El elemento `Action` de una política basada en identidad de IAM describe la acción o las acciones específicas que la política permitirá o denegará. Las acciones políticas suelen tener el mismo nombre que la operación de AWS API asociada. La acción se utiliza en una política para otorgar permisos para realizar la operación asociada.

Las acciones políticas AWS IoT Events utilizan el siguiente prefijo antes de la acción: `iotevents:`. Por ejemplo, para conceder permiso a alguien para crear una AWS IoT Events entrada con la operación de la AWS IoT Events `CreateInput` API, debes incluir la `iotevents:CreateInput` acción en su política. Para conceder permiso a alguien para enviar una entrada con la operación de la AWS IoT Events `BatchPutMessage` API, debes incluir la `iotevents-data:BatchPutMessage` acción en su política. Las declaraciones de política deben incluir un `NotAction` elemento `Action` o. AWS IoT Events define su propio conjunto de acciones que describen las tareas que puede realizar con este servicio.

Para especificar varias acciones en una única instrucción, sepárelas con comas del siguiente modo:

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

Puede utilizar caracteres comodín para especificar varias acciones (*). Por ejemplo, para especificar todas las acciones que comiencen con la palabra `Describe`, incluya la siguiente acción:

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

Para ver una lista de AWS IoT Events acciones, consulte las [acciones definidas por AWS IoT Events](#) en la Guía del usuario de IAM.

Recursos

El elemento `Resource` especifica el objeto u objetos a los que se aplica la acción. Las instrucciones deben contener un elemento `Resource` o `NotResource`. Especifique un recurso con un ARN o el carácter comodín (*) para indicar que la instrucción se aplica a todos los recursos.

El recurso del modelo de AWS IoT Events detector tiene el siguiente ARN:

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

Para obtener más información sobre el formato de ARNs, consulte [Identificar AWS los recursos con los nombres de los recursos de Amazon \(ARNs\)](#).

Por ejemplo, para especificar el modelo de detector `Foobar` en su instrucción, utilice el siguiente ARN:

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

Para especificar todas las instancias que pertenecen a una cuenta específica, utilice el carácter comodín (*):

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

Algunas AWS IoT Events acciones, como las de creación de recursos, no se pueden realizar en un recurso específico. En dichos casos, debe utilizar el carácter comodín (*).

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

Algunas acciones AWS IoT Events de la API implican varios recursos. Por ejemplo, `CreateDetectorModel` hace referencia a las entradas en sus declaraciones de condición, por lo

que un usuario debe tener permisos para utilizar la entrada y el modelo de detector. Para especificar varios recursos en una sola sentencia, sepárelos ARNs con comas.

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

Para ver una lista de los tipos de AWS IoT Events recursos y sus respectivos tipos ARNs, consulte [los recursos definidos por AWS IoT Events](#) en la Guía del usuario de IAM. Para obtener información sobre las acciones con las que puede especificar el ARN de cada recurso, consulte [Acciones definidas por AWS IoT Events](#).

Claves de condición

El elemento Condition (o bloque de Condition) permite especificar condiciones en las que entra en vigor una instrucción. El elemento Condition es opcional. Puede crear expresiones condicionales que utilizan [operadores de condición](#), tales como igual o menor que, para que coincida la condición de la política con valores de la solicitud.

Si especifica varios elementos de Condition en una instrucción o varias claves en un único elemento de Condition, AWS las evalúa mediante una operación AND lógica. Si especifica varios valores para una única clave de condición, AWS evalúa la condición mediante una operación lógicaOR. Se deben cumplir todas las condiciones antes de que se concedan los permisos de la instrucción.

También puedes utilizar variables de marcador de posición al especificar condiciones. Por ejemplo, puede conceder a un usuario permiso para acceder a un recurso solo si está etiquetado con su nombre de usuario. Para obtener más información, consulte [Elementos de la política de IAM: Variables y etiquetas](#) en la Guía del usuario de IAM.

AWS IoT Events no proporciona ninguna clave de condición específica del servicio, pero admite el uso de algunas claves de condición globales. Para ver todas las claves de condición AWS globales, consulte las claves de [contexto de condición AWS globales](#) en la Guía del usuario de IAM».

Ejemplos

Para ver ejemplos de políticas AWS IoT Events basadas en la identidad, consulte. [AWS IoT Events ejemplos de políticas basadas en la identidad](#)

Políticas de AWS IoT Events basadas en recursos

AWS IoT Events no admite políticas basadas en recursos». Para ver un ejemplo de una página detallada de política basada en recursos, consulte <https://docs.aws.amazon.com/lambda/latest/dg/access-control-resource-based.html>.

Autorización basada en etiquetas de AWS IoT Events

Puedes adjuntar etiquetas a AWS IoT Events los recursos o pasarles etiquetas en una solicitud. AWS IoT Events Para controlar el acceso en función de etiquetas, debe proporcionar información de las etiquetas en el [elemento de condición](#) de una política utilizando las claves de condición `iotevents:ResourceTag/key-name`, `aws:RequestTag/key-name` o `aws:TagKeys`. Para obtener más información acerca del etiquetado de recursos de AWS IoT Events , consulte [Etiquetar sus recursos AWS IoT Events](#).

Para consultar un ejemplo de política basada en la identidad para limitar el acceso a un recurso en función de las etiquetas de ese recurso, consulte [Vea AWS IoT Events las entradas en función de las etiquetas](#).

AWS IoT Events Funciones de IAM

Un [rol de IAM](#) es una entidad dentro de usted Cuenta de AWS que tiene permisos específicos.

Utilizar credenciales temporales con AWS IoT Events

Puede utilizar credenciales temporales para iniciar sesión con federación, asumir un rol de IAM o asumir un rol de acceso entre cuentas. Las credenciales de seguridad temporales se obtienen llamando a AWS Security Token Service (AWS STS) operaciones de la API, como [AssumeRole](#) o [GetFederationToken](#).

AWS IoT Events no admite el uso de credenciales temporales.

Roles vinculados a servicios

Los [roles vinculados a un servicio](#) permiten a AWS los servicios acceder a los recursos de otros servicios para completar una acción en tu nombre. Los roles vinculados a servicios aparecen en la cuenta de IAM y son propiedad del servicio. Un administrador de IAM puede ver, pero no editar, los permisos de los roles vinculados a servicios.

AWS IoT Events no admite funciones vinculadas al servicio.

Roles de servicio

Esta característica permite que un servicio asuma un [rol de servicio](#) en su nombre. Este rol permite que el servicio obtenga acceso a los recursos de otros servicios para completar una acción en su nombre. Los roles de servicio aparecen en su cuenta de IAM y son propiedad de la cuenta. Esto significa que un administrador de IAM puede cambiar los permisos de este rol. Sin embargo, hacerlo podría deteriorar la funcionalidad del servicio.

AWS IoT Events admite funciones de servicio.

AWS IoT Events ejemplos de políticas basadas en la identidad

De forma predeterminada, los usuarios y los roles no tienen permiso para crear o modificar AWS IoT Events recursos. Tampoco pueden realizar tareas con la AWS API AWS Management Console AWS CLI, o. Un administrador de IAM debe crear políticas de IAM que concedan permisos a los usuarios y a los roles para realizar operaciones de la API concretas en los recursos especificados que necesiten. El administrador debe asociar esas políticas a los usuarios o grupos que necesiten esos permisos.

Para obtener más información acerca de cómo crear una política basada en identidad de IAM con estos documentos de políticas de JSON de ejemplo, consulte [Creación de políticas en la pestaña JSON](#) en la Guía del usuario de IAM.

Temas

- [Prácticas recomendadas relativas a políticas](#)
- [Mediante la consola de AWS IoT Events](#)
- [Permite a los usuarios ver sus propios permisos en AWS IoT Events](#)
- [Acceda a una entrada AWS IoT Events](#)
- [Vea AWS IoT Events las entradas en función de las etiquetas](#)

Prácticas recomendadas relativas a políticas

Las políticas basadas en identidad son muy eficaces. Determinan si alguien puede crear AWS IoT Events recursos de tu cuenta, acceder a ellos o eliminarlos. Estas acciones pueden generar costos adicionales para su Cuenta de AWS. Siga estas directrices y recomendaciones al crear o editar políticas basadas en identidades:

- Comience a utilizar las políticas AWS gestionadas: para empezar a AWS IoT Events utilizarlas rápidamente, utilice las políticas AWS gestionadas para conceder a sus empleados los permisos que necesitan. Estas políticas ya están disponibles en su cuenta, y las mantiene y actualiza AWS. Para obtener más información, consulte [Cómo empezar a usar permisos con políticas AWS administradas](#) en la Guía del usuario de IAM.
- Conceder privilegios mínimos: al crear políticas personalizadas, conceda solo los permisos necesarios para llevar a cabo una tarea. Comience con un conjunto mínimo de permisos y conceda permisos adicionales según sea necesario. Por lo general, es más seguro que comenzar con permisos demasiado tolerantes e intentar hacerlos más estrictos más adelante. Para obtener más información, consulte [Conceder privilegios mínimos](#) en la Guía del usuario de IAM.
- Habilitar MFA para operaciones confidenciales: para mayor seguridad, obligue a los usuarios a que utilicen la autenticación multifactor (MFA) para acceder a recursos u operaciones de API confidenciales. Para obtener más información, consulte [Uso de la autenticación multifactor \(MFA\) en AWS](#) en la Guía del usuario de IAM.
- Utilizar condiciones de política para mayor seguridad: en la medida en que sea práctico, defina las condiciones en las que sus políticas basadas en identidad permitan el acceso a un recurso. Por ejemplo, puede escribir condiciones para especificar un rango de direcciones IP permitidas desde el que debe proceder una solicitud. También puede escribir condiciones para permitir solicitudes solo en un intervalo de hora o fecha especificado o para solicitar el uso de SSL o MFA. Para obtener más información, consulte [Elementos de la política JSON de IAM: condición](#) en la Guía del usuario de IAM.

Mediante la consola de AWS IoT Events

Para acceder a la AWS IoT Events consola, debe tener un conjunto mínimo de permisos. Estos permisos deben permitirle enumerar y ver detalles sobre los AWS IoT Events recursos de su cuenta Cuenta de AWS. Si crea una política basada en identidades que sea más restrictiva que el mínimo de permisos necesarios, la consola no funcionará del modo esperado para las entidades (usuarios o roles) que tengan esa política.

Para garantizar que esas entidades puedan seguir utilizando la AWS IoT Events consola, adjunte también la siguiente política AWS gestionada a las entidades. Para obtener más información, consulte [Agregar de permisos a un usuario](#) en la Guía del usuario de IAM:

```
{
  "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}"
}
]
}

```

No es necesario que concedas permisos mínimos de consola a los usuarios que solo realizan llamadas a la API AWS CLI o a la AWS API. En su lugar, permite acceso únicamente a las acciones que coincidan con la operación de API que intenta realizar.

Permite a los usuarios ver sus propios permisos en AWS IoT Events

En este ejemplo, se muestra cómo podría crear una política que permita a los usuarios ver las políticas administradas e insertadas que se asocian a la identidad de sus usuarios. Permitir a los usuarios ver sus propios permisos de IAM es útil para mejorar la seguridad y las capacidades de

autoservicio. Esta política incluye permisos para completar esta acción en la consola o mediante programación mediante la API o. AWS CLI AWS

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "ViewOwnUserInfo",
      "Effect": "Allow",
      "Action": [
        "iam:GetUserPolicy",
        "iam:ListGroupForUser",
        "iam:ListAttachedUserPolicies",
        "iam:ListUserPolicies",
        "iam:GetUser"
      ],
      "Resource": [
        "arn:aws:iam::*:user/${aws:username}"
      ]
    },
    {
      "Sid": "NavigateInConsole",
      "Effect": "Allow",
      "Action": [
        "iam:GetGroupPolicy",
        "iam:GetPolicyVersion",
        "iam:GetPolicy",
        "iam:ListAttachedGroupPolicies",
        "iam:ListGroupPolicies",
        "iam:ListPolicyVersions",
        "iam:ListPolicies",
        "iam:ListUsers"
      ],
      "Resource": "*"
    }
  ]
}
```

Acceda a una entrada AWS IoT Events

El control detallado del acceso a AWS IoT Events las entradas es importante para mantener la seguridad en entornos multiusuario o multiequipo. En esta sección se muestra cómo crear políticas

de IAM que concedan acceso a AWS IoT Events entradas específicas y, al mismo tiempo, restrinjan el acceso a otras.

En este ejemplo, puede conceder a un usuario el Cuenta de AWS acceso a una de sus AWS IoT Events entradas, `exampleInput`. También puedes permitir que el usuario añada, actualice y elimine entradas.

La política concede los permisos `iotevents:ListInputs`, `iotevents:DescribeInput`, `iotevents:CreateInput`, `iotevents>DeleteInput` y `iotevents:UpdateInput` al usuario. Para ver un ejemplo de tutorial del Amazon Simple Storage Service (Amazon S3) que concede permisos a los usuarios y los prueba mediante la consola, [consulte Controlar el acceso a un bucket con políticas de usuario](#).

```
{
  "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/*"
  }
]
}

```

Vea AWS IoT Events las entradas en función de las etiquetas

Las etiquetas ayudan a organizar AWS IoT Events los recursos. Puede utilizar condiciones en su política basada en la identidad para controlar el acceso a AWS IoT Events los recursos en función de las etiquetas. En este ejemplo se muestra cómo puede crear una política que permita ver un *input*. Sin embargo, los permisos solo se conceden si el Owner de la etiqueta *input* tiene el valor del nombre de usuario de dicho usuario. Esta política también proporciona los permisos necesarios para llevar a cabo esta acción en la consola.

```

{
  "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}"}
      }
    }
  ]
}

```

También puede adjuntar esta política al usuario de en su cuenta. Si un usuario llamado richard-roe intenta ver una AWS IoT Events *input*, *input* debe estar etiquetada Owner=richard-roe owner=richard-roe. De lo contrario, se le deniega el acceso. La clave de la etiqueta de condición Owner coincide con los nombres de las claves de condición Owner y owner porque no distinguen entre mayúsculas y minúsculas. Para obtener más información, consulta [Elementos de la política de JSON de IAM: Condición](#) en la Guía del usuario de IAM.

Prevención policial confusa entre servicios para AWS IoT Events

Note

- El AWS IoT Events servicio solo permite usar roles para iniciar acciones en la misma cuenta en la que se creó un recurso. Esto ayuda a evitar que un diputado confuso ataque AWS IoT Events.
- Esta página le servirá de referencia para ver cómo funciona el confuso problema de los diputados y puede evitarse en caso de que el AWS IoT Events servicio permita el uso de recursos entre cuentas cruzadas.

El problema de la sustitución confusa es un problema de seguridad en el que una entidad que no tiene permiso para realizar una acción puede obligar a una entidad con más privilegios a realizar la acción. En AWS, la suplantación de identidad entre servicios puede provocar el confuso problema de un diputado.

La suplantación entre servicios puedes producirse cuando un servicio (el servicio que lleva a cabo las llamadas) llama a otro servicio (el servicio al que se llama). El servicio que lleva a cabo las llamadas se puedes manipular para utilizar sus permisos a fin de actuar en función de los recursos de otro cliente de una manera en la que no debe tener permiso para acceder. Para evitarlo, AWS proporciona herramientas que le ayudan a proteger los datos de todos los servicios cuyos directores de servicio tengan acceso a los recursos de su cuenta.

Recomendamos utilizar las claves de contexto de condición [aws:SourceAccount](#) global [aws:SourceArn](#) y las claves contextuales en las políticas de recursos para limitar los permisos que se AWS IoT Events otorgan a otro servicio al recurso. Si el valor de `aws:SourceArn` no contiene el ID de cuenta, como un ARN de bucket de Amazon S3, debe utilizar ambas claves de contexto de condición global para limitar los permisos. Si utiliza claves de contexto de condición global y el valor de `aws:SourceArn` contiene el ID de cuenta, el valor de `aws:SourceAccount` y la cuenta en el valor de `aws:SourceArn` deben utilizar el mismo ID de cuenta cuando se utiliza en la misma instrucción de política.

Utiliza `aws:SourceArn` si desea que solo se asocie un recurso al acceso entre servicios. Utiliza `aws:SourceAccount` si quiere permitir que cualquier recurso de esa cuenta se asocie al uso entre servicios. El valor de `aws:SourceArn` debe ser el modelo de detector o el modelo de alarma asociado a la solicitud `sts:AssumeRole`.

La forma más eficaz de protegerse contra el problema de la sustitución confusa es utilizar la clave de contexto de condición global de `aws:SourceArn` con el ARN completo del recurso. Si no conoce el ARN completo del recurso o si especifica varios recursos, utiliza la clave de condición de contexto global `aws:SourceArn` con comodines (*) para las partes desconocidas del ARN. Por ejemplo, `arn:aws:iotevents*:123456789012:*`.

En los ejemplos siguientes se muestra cómo utilizar las claves de contexto de condición `aws:SourceAccount` global `aws:SourceArn` y las claves contextuales AWS IoT Events para evitar el confuso problema de los diputados.

Temas

- [Ejemplo: acceso seguro a un modelo AWS IoT Events de detector](#)
- [Ejemplo: acceso seguro a un modelo AWS IoT Events de alarma](#)
- [Ejemplo: acceder a un AWS IoT Events recurso en una región específica](#)
- [Ejemplo: configurar las opciones de registro para AWS IoT Events](#)

Ejemplo: acceso seguro a un modelo AWS IoT Events de detector

En este ejemplo se muestra cómo crear una política de IAM que conceda acceso de forma segura a un modelo de detector específico en AWS IoT Events. La política utiliza condiciones para garantizar que solo la AWS cuenta y el AWS IoT Events servicio especificados puedan asumir la función, lo que añade un nivel adicional de seguridad. En este ejemplo, el rol solo puede acceder al modelo de detector mencionado `WindTurbine01`.

```
{
  "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/WindTurbine01"
    }
  }
}
]
}

```

Ejemplo: acceso seguro a un modelo AWS IoT Events de alarma

Este ejemplo demuestra cómo crear una política de IAM que permita acceder de forma segura AWS IoT Events a los modelos de alarma. La política utiliza condiciones para garantizar que solo la AWS cuenta y el AWS IoT Events servicio especificados puedan asumir la función.

En este ejemplo, el rol puede acceder a cualquier modelo de alarma de la AWS cuenta especificada, como lo indica el * comodín en el ARN del modelo de alarma. Las `aws:SourceArn` condiciones `aws:SourceAccount` y las dos funcionan juntas para evitar el confuso problema de los diputados.

```

{
  "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/*"
        }
      }
    }
  ]
}

```

Ejemplo: acceder a un AWS IoT Events recurso en una región específica

En este ejemplo, se muestra cómo configurar un rol de IAM para acceder a AWS IoT Events los recursos de una AWS región específica. Al utilizar políticas de IAM específicas para cada región ARNs , puede restringir el acceso a AWS IoT Events los recursos en diferentes áreas geográficas. Este enfoque puede ayudar a mantener la seguridad y el cumplimiento en las implementaciones en varias regiones. La región de este ejemplo es. *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:"
        }
      }
    }
  ]
}
```

Ejemplo: configurar las opciones de registro para AWS IoT Events

El registro adecuado es importante para supervisar, depurar y auditar AWS IoT Events las aplicaciones. En esta sección se proporciona una descripción general de las opciones de registro disponibles en AWS IoT Events.

En este ejemplo, se muestra cómo configurar una función de IAM que AWS IoT Events permita registrar datos en los CloudWatch registros. El uso de caracteres comodín (*) en el ARN del recurso permite un registro completo en toda AWS IoT Events la infraestructura.

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

Solucionar problemas de AWS IoT Events identidad y acceso

Utilice la siguiente información como ayuda para diagnosticar y solucionar los problemas más comunes que pueden surgir al trabajar con un AWS IoT Events IAM.

Temas

- [No estoy autorizado a realizar ninguna acción en AWS IoT Events](#)
- [No tengo autorización para realizar iam:PassRole](#)
- [Quiero permitir que personas ajenas a mí accedan Cuenta de AWS a mis AWS IoT Events recursos](#)

No estoy autorizado a realizar ninguna acción en AWS IoT Events

Si AWS Management Console le indica que no está autorizado a realizar una acción, debe ponerse en contacto con su administrador para obtener ayuda. Su administrador es la persona que le facilitó su nombre de usuario y contraseña.

En el siguiente ejemplo, el error se produce cuando el usuario de IAM, `mateojackson`, intenta utilizar la consola para ver detalles sobre una *input*, pero no tiene permisos `iotevents:ListInputs`.

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

En este caso, Mateo pide a su administrador que actualice sus políticas de forma que pueda obtener acceso al recurso *my-example-input* mediante la acción `iotevents:ListInput`.

No tengo autorización para realizar `iam:PassRole`

Si recibe un error que indica que no tiene autorización para realizar la acción `iam:PassRole`, las políticas deben actualizarse a fin de permitirle pasar un rol a AWS IoT Events.

Algunos Servicios de AWS permiten transferir una función existente a ese servicio en lugar de crear una nueva función de servicio o una función vinculada a un servicio. Para ello, debe tener permisos para transferir el rol al servicio.

En el siguiente ejemplo, el error se produce cuando un usuario de IAM denominado `marymajor` intenta utilizar la consola para realizar una acción en AWS IoT Events. Sin embargo, la acción requiere que el servicio cuente con permisos que otorguen un rol de servicio. Mary no tiene permisos para transferir el rol al servicio.

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

En este caso, las políticas de Mary se deben actualizar para permitirle realizar la acción `iam:PassRole`.

Si necesita ayuda, póngase en contacto con su administrador. AWS El gestor es la persona que le proporcionó las credenciales de inicio de sesión.

Quiero permitir que personas ajenas a mí accedan Cuenta de AWS a mis AWS IoT Events recursos

Puedes crear un rol que los usuarios de otras cuentas o las personas externas a la organización puedan utilizar para acceder a sus recursos. Puedes especificar una persona de confianza para que

asuma el rol. En el caso de los servicios que respaldan las políticas basadas en recursos o las listas de control de acceso (ACLs), puedes usar esas políticas para permitir que las personas accedan a tus recursos.

Consulte los siguientes temas para determinar sus mejores opciones:

- Para saber si AWS IoT Events es compatible con estas funciones, consulte [¿Cómo AWS IoT Events funciona con IAM.](#)
- Para obtener información sobre cómo proporcionar acceso a los recursos de su Cuentas de AWS propiedad, consulte [Proporcionar acceso a un usuario de IAM en otro usuario de su propiedad Cuenta de AWS en](#) la Guía del usuario de IAM.
- Para obtener información sobre cómo proporcionar acceso a tus recursos a terceros Cuentas de AWS, consulta [Cómo proporcionar acceso a recursos que Cuentas de AWS son propiedad de terceros](#) en la Guía del usuario de IAM.
- Para obtener información sobre cómo proporcionar acceso mediante una federación de identidades, consulta [Proporcionar acceso a usuarios autenticados externamente \(identidad federada\)](#) en la Guía del usuario de IAM.
- Para conocer sobre la diferencia entre las políticas basadas en roles y en recursos para el acceso entre cuentas, consulte [Acceso a recursos entre cuentas en IAM](#) en la Guía del usuario de IAM.

Supervisión AWS IoT Events para mantener la confiabilidad, la disponibilidad y el rendimiento

La supervisión es una parte importante del mantenimiento de la confiabilidad, la disponibilidad y el rendimiento de AWS IoT Events sus AWS soluciones. Debe recopilar los datos de supervisión de todas las partes de la AWS solución para poder depurar con mayor facilidad una falla multipunto en caso de que se produzca. Antes de empezar a monitorizar AWS IoT Events, debe crear un plan de monitorización que incluya respuestas a las siguientes preguntas:

- ¿Cuáles son los objetivos de la monitorización?
- ¿Qué recursos va a monitorizar?
- ¿Con qué frecuencia va a monitorizar estos recursos?
- ¿Qué herramientas de monitorización va a utilizar?
- ¿Quién se encargará de realizar las tareas de supervisión?
- ¿Quién debería recibir una notificación cuando surjan problemas?

El siguiente paso es establecer una línea base para el AWS IoT Events rendimiento normal de su entorno, midiendo el rendimiento en distintos momentos y bajo diferentes condiciones de carga. A medida que monitoree AWS IoT Events, almacene los datos de monitorización históricos para que pueda compararlos con los datos de rendimiento actual, identificar los patrones de rendimiento normal y las anomalías en el rendimiento, así como desarrollar métodos para la resolución de problemas.

Por ejemplo, si utilizas Amazon EC2, puedes supervisar el uso de la CPU, la E/S del disco y el uso de la red de tus instancias. Si el desempeño no alcanza los valores del punto de referencia establecido, es posible que deba volver a configurar u optimizar la instancia para reducir la utilización de la CPU, mejorar la E/S de disco o reducir el tráfico de red.

Temas

- [Herramientas disponibles para monitorear AWS IoT Events](#)
- [Monitorización AWS IoT Events con Amazon CloudWatch](#)
- [Registrar llamadas a AWS IoT Events la API con AWS CloudTrail](#)

Herramientas disponibles para monitorear AWS IoT Events

AWS proporciona varias herramientas que puede utilizar para supervisar AWS IoT Events. Puede configurar algunas de estas herramientas para que monitoricen por usted, pero otras herramientas requieren intervención manual. Le recomendamos que automatice las tareas de monitorización en la medida de lo posible.

Herramientas de monitoreo automatizadas

Puede utilizar las siguientes herramientas de supervisión automatizadas para observar AWS IoT Events e informar cuando algo va mal:

- Amazon CloudWatch Logs: supervise, almacene y acceda a sus archivos de registro desde AWS CloudTrail u otras fuentes. Para obtener más información, consulta [Uso de los CloudWatch paneles de Amazon](#) en la Guía del CloudWatch usuario de Amazon.
- AWS CloudTrail Supervisión de registros: comparta archivos de registro entre cuentas, supervise los archivos de CloudTrail registro en tiempo real enviándolos a CloudWatch Logs, cree aplicaciones de procesamiento de registros en Java y valide que sus archivos de registro no hayan cambiado después de su entrega. CloudTrail Para obtener más información, consulte [Trabajar con archivos de CloudTrail registro](#) en la Guía del AWS CloudTrail usuario.

Herramientas de supervisión manuales

Otra parte importante del monitoreo AWS IoT Events consiste en monitorear manualmente los elementos que las CloudWatch alarmas no cubren. El AWS IoT Events panel de control y otros paneles de la AWS consola proporcionan una at-a-glance vista del estado de su AWS entorno. CloudWatch Le recomendamos que compruebe también los archivos de registro. AWS IoT Events

- La AWS IoT Events consola muestra:
 - Modelos de detector
 - Detectores
 - Entradas
 - Configuración
- La página de CloudWatch inicio muestra:
 - Alarmas y estado actual
 - Gráficos de alarmas y recursos
 - Estado de los servicios

Además, puede CloudWatch hacer lo siguiente:

- Crear: [crear un CloudWatch panel de control](#) para monitorear los servicios que le interesan
- Realizar un gráfico con los datos de las métricas para resolver problemas y descubrir tendencias
- Busque y explore todas las métricas AWS de sus recursos
- Crear y editar las alarmas de notificación de problemas

Monitorización AWS IoT Events con Amazon CloudWatch

Al desarrollar o depurar un modelo de AWS IoT Events detector, necesita saber lo que AWS IoT Events está haciendo y los errores que encuentra. Amazon CloudWatch monitorea tus AWS recursos y las aplicaciones en las que AWS ejecutas en tiempo real. De este CloudWatch modo, obtiene visibilidad en todo el sistema sobre el uso de los recursos, el rendimiento de las aplicaciones y el estado operativo. [Habilite el CloudWatch registro de Amazon al desarrollar modelos AWS IoT Events de detectores](#) contiene información sobre cómo habilitar el CloudWatch registro de AWS IoT Events. Para generar registros como el que se muestra a continuación, debe establecer el nivel de verbosidad en «Depurar» y proporcionar uno o más objetivos de depuración, con un nombre de modelo de detector y uno opcional. KeyValue

El siguiente ejemplo muestra una entrada de registro de nivel CloudWatch DEBUG generada por 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
  }
]
```

Registrar llamadas a AWS IoT Events la API con AWS CloudTrail

AWS IoT Events está integrado con AWS CloudTrail un servicio que proporciona un registro de las acciones realizadas por un usuario, un rol o un AWS servicio en AWS IoT Events. CloudTrail captura todas las llamadas a la API AWS IoT Events como eventos, incluidas las llamadas desde la AWS IoT Events consola y desde las llamadas de código a AWS IoT Events APIs.

Si crea una ruta, puede habilitar la entrega continua de CloudTrail eventos a un bucket de Amazon S3, incluidos los eventos para AWS IoT Events. Si no configura una ruta, podrá ver los eventos más recientes en la CloudTrail consola, en el historial de eventos. Con la información recopilada por usted CloudTrail, puede determinar el destinatario de la solicitud AWS IoT Events, la dirección IP desde la que se realizó la solicitud, quién la realizó, cuándo se realizó y detalles adicionales.

Para obtener más información CloudTrail, consulte la [Guía AWS CloudTrail del usuario](#).

AWS IoT Events información en CloudTrail

CloudTrail está habilitada en su AWS cuenta al crear la cuenta. Cuando se produce una actividad en AWS IoT Events, esa actividad se registra en un CloudTrail evento junto con otros eventos de AWS servicio en el historial de eventos. Puedes ver, buscar y descargar los eventos recientes en tu AWS cuenta. Para obtener más información, consulte [Trabajar con el historial de CloudTrail eventos](#).

Para tener un registro continuo de los eventos de tu AWS cuenta, incluidos los eventos de tu cuenta AWS IoT Events, crea una ruta. Un rastro permite CloudTrail entregar archivos de registro a un bucket de Amazon S3. De forma predeterminada, cuando crea una ruta en la consola, la ruta se aplica a todas AWS las regiones. La ruta registra los eventos de todas las regiones de la AWS partición y envía los archivos de registro al bucket de Amazon S3 que especifique. Además, puede configurar otros AWS servicios para analizar más a fondo los datos de eventos recopilados en los CloudTrail registros y actuar en función de ellos. Para obtener más información, consulte:

- [Crear una ruta para tu AWS cuenta](#)
- [Servicios e integraciones compatibles con CloudTrail](#)

- [Configuración de las notificaciones de Amazon SNS para CloudTrail](#)
- [Recibir archivos de CloudTrail registro de varias regiones](#) y [recibir archivos de CloudTrail registro de varias cuentas](#)

Cada entrada de registro o evento contiene información sobre quién generó la solicitud. La información de identidad del usuario lo ayuda a determinar lo siguiente:

- Si la solicitud se realizó con las credenciales raíz o del usuario de IAM.
- Si la solicitud se realizó con credenciales de seguridad temporales de un rol o fue un usuario federado.
- Si la solicitud la realizó otro AWS servicio.

Para obtener más información, consulte el elemento [CloudTrailUserIdentity](#). AWS IoT Events [las acciones se documentan en la referencia de la API AWS IoT Events](#).

Descripción de las entradas de los archivos de AWS IoT Events registro

Un rastro es una configuración que permite la entrega de eventos como archivos de registro a un bucket de Amazon S3 que usted especifique. AWS CloudTrail Los archivos de registro contienen una o más entradas de registro. Un evento representa una solicitud única de cualquier fuente e incluye información sobre la acción solicitada, la fecha y la hora de la acción, los parámetros de la solicitud, etc. CloudTrail Los archivos de registro no son un seguimiento ordenado de las llamadas a la API pública, por lo que no aparecen en ningún orden específico.

Cuando se habilita el CloudTrail registro en tu AWS cuenta, la mayoría de las llamadas a la API realizadas a AWS IoT Events las acciones se registran en los archivos de CloudTrail registro, donde se escriben junto con otros registros de AWS servicio. CloudTrail determina cuándo crear y escribir en un nuevo archivo en función del período de tiempo y del tamaño del archivo.

Cada entrada de registro contiene información sobre quién generó la solicitud. La información de identidad del usuario en la entrada de registro le ayuda a determinar lo siguiente:

- Si la solicitud se realizó con las credenciales raíz o del usuario de IAM.
- Si la solicitud se realizó con credenciales de seguridad temporales de un rol o fue un usuario federado.
- Si la solicitud la realizó otro AWS servicio.

Puede almacenar sus archivos de registro en su bucket de Amazon S3 durante todo el tiempo que desee, pero también puede definir reglas de ciclo de vida de Amazon S3 para archivar o eliminar archivos de registro automáticamente. De forma predeterminada, los archivos de registro se cifran con el cifrado del servidor (SSE) de Amazon S3.

Para recibir una notificación cuando se entreguen los archivos de registro, puede CloudTrail configurar la publicación de notificaciones de Amazon SNS cuando se entreguen nuevos archivos de registro. Para obtener más información, consulte [Configuración de notificaciones de Amazon SNS para CloudTrail](#).

También puede AWS IoT Events agrupar archivos de registro de varias AWS regiones y AWS cuentas en un único bucket de Amazon S3.

Para obtener más información, consulte [Recepción de archivos de CloudTrail registro de varias regiones](#) y [Recepción de archivos de CloudTrail registro de varias cuentas](#).

Ejemplo: DescribeDetector acción para CloudTrail

El siguiente ejemplo muestra una entrada de CloudTrail registro que demuestra la DescribeDetector acción.

```
{
  "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"
}

```

Ejemplo: CreateDetectorModel acción para CloudTrail

El siguiente ejemplo muestra una entrada de CloudTrail registro que demuestra la CreateDetectorModel acción.

```

{
  "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"
}

```

Ejemplo: CreateInput acción para CloudTrail

El siguiente ejemplo muestra una entrada de CloudTrail registro que demuestra la CreateInput acción.

```

{
  "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-e113a7bde1e",
"eventType": "AwsApiCall",
"recipientAccountId": "123456789012"
}

```

Ejemplo: DeleteDetectorModel acción para CloudTrail

El siguiente ejemplo muestra una entrada de CloudTrail registro que demuestra la DeleteDetectorModel acción.

```

{
  "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"
}

```

Ejemplo: DeleteInput acción para CloudTrail

El siguiente ejemplo muestra una entrada de CloudTrail registro que demuestra la DeleteInput acción.

```

{
  "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"
}

```

Ejemplo: DescribeDetectorModel acción para CloudTrail

El siguiente ejemplo muestra una entrada de CloudTrail registro que demuestra la DescribeDetectorModel acción.

```

{
  "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"
  }
}

```

Ejemplo: DescribeInput acción para CloudTrail

El siguiente ejemplo muestra una entrada de CloudTrail registro que demuestra la DescribeInput acción.

```

{
  "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"
}

```

Ejemplo: DescribeLoggingOptions acción para CloudTrail

El siguiente ejemplo muestra una entrada de CloudTrail registro que demuestra la DescribeLoggingOptions acción.

```

{
    "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"
}

```

Ejemplo: ListDetectorModels acción para CloudTrail

El siguiente ejemplo muestra una entrada de CloudTrail registro que demuestra la ListDetectorModels acción.

```

{
  "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": "CkZEZXRlY3Rvck1vZGVsMl9saXN0ZGV0ZWNoY3Jtb2RlbnHN0ZXN0X2VlOWJkZTk1YT",
    "maxResults": 3
  },
  "responseElements": null,
  "requestID": "6d70f262-da95-4bb5-94b4-c08369df75bb",
  "eventID": "2d01a25c-d5c7-4233-99fe-ce1b8ec05516",
  "eventType": "AwsApiCall",
  "recipientAccountId": "123456789012"
}

```

Ejemplo: ListDetectorModelVersions acción para CloudTrail

El siguiente ejemplo muestra una entrada de CloudTrail registro que demuestra la ListDetectorModelVersions acción.

```
{
  "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": "e6ecb277-6bd8-44ea-8abd-fbf40ac044ee",
  "eventID": "fc6281a2-3fac-4e1e-98e0-ca6560b8b8be",
  "eventType": "AwsApiCall",
  "recipientAccountId": "123456789012"
}
```

Ejemplo: ListDetectors acción para CloudTrail

El siguiente ejemplo muestra una entrada de CloudTrail registro que demuestra la ListDetectors acción.

```
{
  "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"
}
```

```
}
```

Ejemplo: ListInputs acción para CloudTrail

El siguiente ejemplo muestra una entrada de CloudTrail registro que demuestra la ListInputs acción.

```
{
  "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"
      }
    },
    "sessionArn": "arn:aws:sts::123456789012:assumed-role/IotEventsLambda-RoleForIotEvents-ABCD123DEF456/IotEvents-EventsLambda"
  },
  "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": "CkhjYW5hcnlfldGVzdF9pbmB1dF9saXN0ZGV0ZWN0b3Jtb2R1bHN0ZXN0ZDU3OGZ",
    "maxResults": 3
  },
  "responseElements": null,
  "requestID": "dd6762a1-1f24-4e63-a986-5ea3938a03da",
}
```

```
"eventID": "c500f6d8-e271-4366-8f20-da4413752469",
"eventType": "AwsApiCall",
"recipientAccountId": "123456789012"
}
```

Ejemplo: PutLoggingOptions acción para CloudTrail

El siguiente ejemplo muestra una entrada de CloudTrail registro que demuestra la PutLoggingOptions acción.

```
{
  "eventVersion": "1.05",
  "userIdentity": {
    "type": "AssumedRole",
    "principalId": "AKIAI44QH8DHBEXAMPLE",
    "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"
      }
    },
    "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"
}

```

Ejemplo: UpdateDetectorModel acción para CloudTrail

El siguiente ejemplo muestra una entrada de CloudTrail registro que demuestra la UpdateDetectorModel acción.

```

{
  "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"
}

```

Ejemplo: UpdateInput acción para CloudTrail

El siguiente ejemplo muestra una entrada de CloudTrail registro que demuestra la UpdateInput acción.

```

{
  "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"
}

```

Ejemplo: BatchPutMessage acción para CloudTrail

AWS IoT Events puede usar una CloudTrail integración para el registro de la API del plano de datos. En este ejemplo, se añaden detalles sobre los eventos de datos a través de la BatchPutMessage acción.

```

{
  "eventVersion": "1.09",
  "userIdentity": {
    "type": "AssumedRole",
    "principalId": "AKIAI44QH8DHBEXAMPLE:PrincipalId",
    "arn": "arn:aws:sts::123456789012:assumed-role/my-iam-role/my-iam-role-entity",
    "accountId": "123456789012",
    "accessKeyId": "AKIAI44QH8DHBEXAMPLE",
    "sessionContext": {
      "sessionIssuer": {
        "type": "Role",
        "principalId": "AKIAI44QH8DHBEXAMPLE",
        "arn": "arn:aws:iam::123456789012:role/my-iam-role",
        "accountId": "123456789012",
        "userName": "sample_user_name"
      },
      "attributes": {

```

```

        "creationDate": "2024-11-22T18:32:41Z",
        "mfaAuthenticated": "false"
    }
},
"eventTime": "2024-11-22T18:57:35Z",
"eventSource": "iotevents.amazonaws.com",
"eventName": "BatchPutMessage",
"awsRegion": "us-east-1",
"sourceIPAddress": "3.239.107.128",
"userAgent": "aws-internal/3",
"requestParameters": {
    "messages": [
        {
            "messageId": "e306d827-b2e4-4439-9c86-411d4242a397",
            "payload": "HIDDEN_DUE_TO_SECURITY_REASONS",
            "inputName": "my_input_name"
        }
    ]
},
"responseElements": {
    "batchPutMessageErrorEntries": []
},
"requestID": "cefc6b63-9ccf-4e31-9177-4aec8e701bfe",
"eventID": "b994b52c-6011-4e3c-ad5f-e784e732fde0",
"readOnly": false,
"resources": [
    {
        "accountId": "123456789012",
        "type": "AWS::IoTEvents::Input",
        "ARN": "arn:aws:iotevents:us-east-1:123456789012:input/
my_input_name"
    }
],
"eventType": "AwsApiCall",
"managementEvent": false,
"recipientAccountId": "123456789012",
"eventCategory": "Data",
"tlsDetails": {
    "tlsVersion": "TLSv1.3",
    "cipherSuite": "TLS_AES_128_GCM_SHA256",
    "clientProvidedHostHeader": "iotevents.us-east-1.amazonaws.com"
}

```

```
},
```

Validación de conformidad para AWS IoT Events

Para saber si uno Servicio de AWS está dentro del ámbito de aplicación de programas de cumplimiento específicos, consulte [Servicios de AWS Alcance por programa de cumplimiento](#) [Servicios de AWS](#) de cumplimiento y elija el programa de cumplimiento que le interese. Para obtener información general, consulte Programas de [AWS cumplimiento > Programas AWS](#).

Puede descargar informes de auditoría de terceros utilizando AWS Artifact. Para obtener más información, consulte [Descarga de informes en AWS Artifact](#).

Su responsabilidad de cumplimiento al Servicios de AWS utilizarlos viene determinada por la confidencialidad de sus datos, los objetivos de cumplimiento de su empresa y las leyes y reglamentos aplicables. AWS proporciona los siguientes recursos para ayudar con el cumplimiento:

- [Cumplimiento de seguridad y gobernanza](#): en estas guías se explican las consideraciones de arquitectura y se proporcionan pasos para implementar las características de seguridad y cumplimiento.
- [Referencia de servicios válidos de HIPAA](#): muestra una lista con los servicios válidos de HIPAA. No todos Servicios de AWS cumplen con los requisitos de la HIPAA.
- [AWS Recursos de](#) de cumplimiento: esta colección de libros de trabajo y guías puede aplicarse a su industria y ubicación.
- [AWS Guías de cumplimiento para clientes](#): comprenda el modelo de responsabilidad compartida desde el punto de vista del cumplimiento. Las guías resumen las mejores prácticas para garantizar la seguridad Servicios de AWS y orientan los controles de seguridad en varios marcos (incluidos el Instituto Nacional de Estándares y Tecnología (NIST), el Consejo de Normas de Seguridad del Sector de Tarjetas de Pago (PCI) y la Organización Internacional de Normalización (ISO)).
- [Evaluación de los recursos con reglas](#) en la guía para AWS Config desarrolladores: el AWS Config servicio evalúa en qué medida las configuraciones de los recursos cumplen con las prácticas internas, las directrices del sector y las normas.
- [AWS Security Hub](#)— Este Servicio de AWS proporciona una visión completa del estado de su seguridad interior AWS. Security Hub utiliza controles de seguridad para evaluar sus recursos de AWS y comprobar su cumplimiento con los estándares y las prácticas recomendadas del sector de la seguridad. Para obtener una lista de los servicios y controles compatibles, consulta la [Referencia de controles de Security Hub](#).

- [Amazon GuardDuty](#): Servicio de AWS detecta posibles amenazas para sus cargas de trabajo. Cuentas de AWS, contenedores y datos mediante la supervisión de su entorno para detectar actividades sospechosas y maliciosas. GuardDuty puede ayudarlo a cumplir con varios requisitos de conformidad, como el PCI DSS, al cumplir con los requisitos de detección de intrusiones exigidos por ciertos marcos de cumplimiento.
- [AWS Audit Manager](#)— Esto le Servicio de AWS ayuda a auditar continuamente su AWS uso para simplificar la gestión del riesgo y el cumplimiento de las normativas y los estándares del sector.

Resiliencia en AWS IoT Events

La infraestructura AWS global se basa en AWS regiones y zonas de disponibilidad. AWS Las regiones proporcionan varias zonas de disponibilidad físicamente independientes y aisladas que se encuentran conectadas mediante redes con un alto nivel de rendimiento y redundancia, además de baja latencia. Con las zonas de disponibilidad, puedes diseñar y utilizar aplicaciones y bases de datos que realizan una conmutación por error automática entre zonas de disponibilidad sin interrupciones. Las zonas de disponibilidad tienen una mayor disponibilidad, tolerancia a errores y escalabilidad que las infraestructuras tradicionales de centros de datos únicos o múltiples.

Para obtener más información sobre AWS las regiones y las zonas de disponibilidad, consulte [infraestructura AWS global](#).

Seguridad de la infraestructura en AWS IoT Events

Como servicio gestionado, AWS IoT Events está protegido por la seguridad de la red AWS global. Para obtener información sobre los servicios AWS de seguridad y cómo se AWS protege la infraestructura, consulte [Seguridad AWS en la nube](#). Para diseñar su AWS entorno utilizando las mejores prácticas de seguridad de la infraestructura, consulte [Protección de infraestructuras en un marco](#) de buena AWS arquitectura basado en el pilar de la seguridad.

Utiliza las llamadas a la API AWS publicadas para acceder a AWS IoT Events través de la red. Los clientes deben admitir lo siguiente:

- Seguridad de la capa de transporte (TLS). Exigimos TLS 1.2 y recomendamos TLS 1.3.
- Conjuntos de cifrado con confidencialidad directa total (PFS) como DHE (Ephemeral Diffie-Hellman) o ECDHE (Elliptic Curve Ephemeral Diffie-Hellman). La mayoría de los sistemas modernos como Java 7 y posteriores son compatibles con estos modos.

Además, las solicitudes deben estar firmadas mediante un ID de clave de acceso y una clave de acceso secreta que esté asociada a una entidad principal de IAM. También puedes utilizar [AWS Security Token Service](#) (AWS STS) para generar credenciales de seguridad temporales para firmar solicitudes.

AWS cuotas de servicio para AWS IoT Events los recursos

La Referencia general de AWS Guía proporciona las cuotas predeterminadas AWS IoT Events para una AWS cuenta. A menos que se especifique lo contrario, cada cuota es por AWS región. Para obtener más información, consulte [Puntos de conexión AWS IoT Events y cuotas](#) y [Service Quotas de AWS](#) en la Guía de Referencia general de AWS .

Para solicitar un aumento de Service Quotas, envíe un caso de soporte a la consola del [Centro de soporte](#). Para obtener más información, consulte [Solicitud de un aumento de cuota](#) en la Guía del usuario de Service Quotas.

Note

- Todos los nombres de modelos de detector y entradas deben ser únicos en una misma cuenta.
- No puede cambiar los nombres de los modelos de detector ni de las entradas después de haberlos creado.

Etiquetar sus recursos AWS IoT Events

Para administrar y organizar sus modelos de detectores y entradas con mayor facilidad, puede asignar opcionalmente sus propios metadatos en forma de etiquetas a cada uno de estos recursos. En esta sección se describe qué son las etiquetas y cómo crearlas.

Conceptos básicos de etiquetas

Las etiquetas le permiten clasificar sus AWS IoT Events recursos de diferentes maneras, por ejemplo, por propósito, propietario o entorno. Esto es útil cuando se tienen muchos recursos del mismo tipo. Puede identificar rápidamente un recurso específico según las etiquetas que le haya asignado.

Cada etiqueta está formada por una clave y un valor opcional, ambos definidos por el usuario. Por ejemplo, podría definir un conjunto de etiquetas para sus entradas que le ayuden a hacer un seguimiento de los dispositivos que envían estas entradas según su tipo. Le recomendamos que cree un conjunto de claves de etiqueta que cumpla sus necesidades para cada tipo de recurso. Mediante el uso de un conjunto coherente de claves de etiquetas, podrá administrar los recursos de más fácilmente.

Puede buscar y filtrar recursos en función de las etiquetas que añada o aplique, utilizar etiquetas para categorizar y realizar un seguimiento de sus gastos y también utilizar etiquetas para controlar el acceso a sus recursos, tal y como se describe en [Uso de etiquetas con políticas de IAM](#) en la Guía para desarrolladores de AWS IoT .

Para facilitar su uso, el editor de etiquetas AWS Management Console proporciona una forma centralizada y unificada de crear y administrar las etiquetas. Para obtener más información, consulte [Introducción al editor de etiquetas](#) en los AWS recursos de etiquetado y en la Guía del usuario del editor de etiquetas.

También puede trabajar con etiquetas mediante la API AWS CLI y la AWS IoT Events API. Puede asociar etiquetas con modelos de detectores y entradas al crearlas a través del campo "Tags" en los siguientes comandos:

- [CreateDetectorModel](#)
- [CreateInput](#)

Puede añadir, modificar o eliminar etiquetas para recursos existentes que admitan el uso de etiquetas con los siguientes comandos:

- [TagResource](#)
- [ListTagsForResource](#)
- [UntagResource](#)

Puede editar las claves y los valores de las etiquetas y también puede eliminar etiquetas de un recurso en cualquier momento. Puede establecer el valor de una etiqueta como una cadena vacía, pero no puede asignarle un valor nulo. Si añade una etiqueta con la misma clave que una etiqueta existente en ese recurso, el nuevo valor sobrescribirá al antiguo. Si elimina un recurso, también se eliminarán todas las etiquetas que este tenga asociadas.

Para obtener más información, consulta [las prácticas recomendadas para etiquetar AWS](#) los recursos

Restricciones y limitaciones en las etiquetas

Se aplican las siguientes restricciones básicas a las etiquetas:

- Número máximo de etiquetas por recurso: 50
- Longitud máxima de la clave: 127 caracteres Unicode en UTF-8
- Longitud máxima del valor: 255 caracteres Unicode en UTF-8
- Las claves y los valores de las etiquetas distinguen entre mayúsculas y minúsculas.
- No utilice el "aws:" prefijo en los nombres o valores de las etiquetas, ya que está reservado para AWS su uso. Los nombres y valores de etiquetas que tienen este prefijo no se pueden editar ni eliminar. Las etiquetas que tengan este prefijo no cuentan para el límite de etiquetas por recurso.
- Si se utiliza su esquema de etiquetado en múltiples servicios y recursos, recuerde que otros servicios pueden tener otras restricciones sobre caracteres permitidos. Los caracteres generalmente permitidos son: letras, espacios y números representables en UTF-8, además de los siguientes caracteres especiales: + - = . _ : / @.

Uso de etiquetas con políticas de IAM

Puede aplicar permisos de nivel de recurso basados en etiquetas en las políticas de IAM que utiliza con las acciones de la API de AWS IoT Events . Esto le ofrece un mejor control sobre los recursos que un usuario puede crear, modificar o utilizar.

Puede utilizar el elemento `Condition` (también llamado bloque `Condition`) junto con las siguientes claves contextuales de condición y valores en una política de IAM para controlar el acceso del usuario (permiso) en función de las etiquetas de un usuario:

- Utilice `aws:ResourceTag/<tag-key>: <tag-value>` para permitir o denegar acciones de los usuarios en recursos con etiquetas específicas.
- Utilice `aws:RequestTag/<tag-key>: <tag-value>` para exigir (o impedir) el uso de una etiqueta específica al realizar una solicitud de API para crear o modificar un recurso que permita etiquetas.
- Utilice `aws:TagKeys: [<tag-key>, ...]` para exigir (o impedir) el uso de un conjunto de claves de etiquetas al realizar una solicitud de API para crear o modificar un recurso que permita etiquetas.

Note

Las claves contextuales de condición y los valores de una política de IAM se aplican únicamente a las acciones de AWS IoT Events en las que un identificador de un recurso que se puede etiquetar es un parámetro obligatorio.

[Control del acceso mediante etiquetas](#) en la Guía del usuario de AWS Identity and Access Management contiene información adicional sobre el uso de etiquetas. La sección [Referencia de políticas JSON de IAM](#) de esta guía incluye sintaxis, descripciones y ejemplos detallados de los elementos, variables y lógica de evaluación de las políticas JSON de IAM.

La siguiente política de ejemplo aplica dos restricciones basadas en etiquetas. Un usuario restringido por esta política:

- No puede dar a un recurso la etiqueta `"env=prod"` (en el ejemplo, consulte la línea `"aws:RequestTag/env" : "prod"`).

- No puede modificar ni obtener acceso a un recurso que tenga una etiqueta "env=prod" existente (en el ejemplo, consulte la línea "aws:RequestTag/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": "*"
    }
]
```

También puede especificar varios valores de etiqueta para una determinada clave de etiqueta encerrándolos en una lista, como se indica a continuación.

```
"StringEquals" : {
    "aws:ResourceTag/env" : ["dev", "test"]
}
```

Note

Si permite o deniega a los usuarios acceso a recursos en función de etiquetas, debe considerar denegar explícitamente a los usuarios la posibilidad de agregar estas etiquetas o retirarlas de los mismos recursos. De lo contrario, es posible que un usuario eluda sus restricciones y obtenga acceso a un recurso modificando sus etiquetas.

Solución de problemas AWS IoT Events

Esta guía de solución de problemas proporciona soluciones a los problemas más comunes que pueden surgir al utilizarla AWS IoT Events. Examine los temas para identificar y resolver problemas relacionados con la detección de eventos, el acceso a los datos, los permisos, las integraciones de servicios, las configuraciones de los dispositivos y mucho más. Con consejos de solución de problemas para la AWS IoT Events consola, la API, la CLI, los errores, la latencia y las integraciones, esta guía tiene como objetivo resolver rápidamente sus problemas para que pueda crear aplicaciones basadas en eventos confiables y escalables.

Temas

- [AWS IoT Events Problemas y soluciones comunes](#)
- [Solución de problemas de un modelo de detector mediante la ejecución de análisis en AWS IoT Events](#)

AWS IoT Events Problemas y soluciones comunes

Consulte la siguiente sección para solucionar los errores y encontrar posibles soluciones para resolverlos. AWS IoT Events

Errores

- [Errores de creación del modelo de detector](#)
- [Actualizaciones de un modelo de detector eliminado](#)
- [Fallo de activación de la acción \(al cumplirse una condición\)](#)
- [Fallo de activación de la acción \(al superar un umbral\)](#)
- [Uso incorrecto de los estados](#)
- [Mensaje de conexión](#)
- [InvalidRequestException mensaje](#)
- [action.setTimerErrores de Amazon CloudWatch Logs](#)
- [Errores de CloudWatch carga útil de Amazon](#)
- [Tipos de datos incompatibles](#)
- [No se pudo enviar el mensaje a AWS IoT Events](#)

Errores de creación del modelo de detector

Aparecen errores cuando intento crear un modelo de detector.

Solución

Al crear un modelo de detector, debe tener en cuenta las siguientes limitaciones.

- Solo se permite una acción en cada campo `action`.
- La `condition` es obligatoria para `transitionEvents`. Es opcional para los eventos `OnEnter`, `OnInput` y `OnExit`.
- Si el campo `condition` está vacío, el resultado evaluado de la expresión de condición equivale a `true`.
- El resultado evaluado de la expresión de condición debe ser un valor booleano. Si el resultado no es un valor booleano, equivale a `false` y no activa las `actions` ni la transición al `nextState` especificados en el evento.

Para obtener más información, consulte [AWS IoT Events restricciones y limitaciones del modelo de detector](#).

Actualizaciones de un modelo de detector eliminado

He actualizado o eliminado un modelo de detector hace unos minutos, pero sigo recibiendo actualizaciones de estado del antiguo modelo de detector a través de mensajes MQTT o alertas SNS.

Solución

Si actualiza, elimina o vuelve a crear un modelo de detector (consulte [UpdateDetectorModel](#)), pasará un tiempo hasta que se eliminen todas las instancias del detector y se utilice el nuevo modelo. Durante este tiempo, es posible que se sigan procesando entradas mediante las instancias de la versión anterior del modelo de detector. Podría seguir recibiendo alertas definidas por el modelo de detector anterior. Espere al menos siete minutos antes de volver a comprobar la actualización o de reportar un error.

Fallo de activación de la acción (al cumplirse una condición)

El detector no logra activar una acción o una transición a un nuevo estado al cumplirse la condición.

Solución

Compruebe que el resultado evaluado de la expresión condicional del detector sea un valor booleano. Si el resultado no es un valor booleano, equivale a `false` y no activa las `action` ni la transición al `nextState` especificados en el evento. Para obtener más información, consulte [Sintaxis de expresiones condicionales](#).

Fallo de activación de la acción (al superar un umbral)

El detector no activa una acción o una transición de evento cuando la variable en una expresión condicional alcanza un valor especificado.

Solución

Si actualiza `setVariable` para `onInput`, `onEnter`, o `onExit`, el nuevo valor no se utiliza al evaluar cualquier `condition` durante el ciclo de procesamiento actual. En cambio, se utiliza el valor original hasta completarse el ciclo actual. Puede cambiar este comportamiento si configura el parámetro `evaluationMethod` en la definición del modelo de detector. Si `evaluationMethod` se establece en `SERIAL`, las variables se actualizan y las condiciones de eventos se evalúan en el orden en que se definan los eventos. Si `evaluationMethod` se establece en `BATCH` (predeterminado), las variables se actualizan y los eventos se ejecutan solo después de que se evalúen todas las condiciones del evento.

Uso incorrecto de los estados

El detector entra en estados incorrectos cuando intento enviar mensajes a las entradas mediante `BatchPutMessage`.

Solución

Si suele enviar varios mensajes [BatchPutMessage](#) a las entradas, no se garantiza el orden en que se procesan los mensajes o las entradas. Para garantizar el orden, envíe los mensajes uno a la vez y espere cada vez que `BatchPutMessage` confirme el éxito.

Mensaje de conexión

Recibo un error (`'Connection aborted.'`, `error(54, 'Connection reset by peer')`) cuando intento llamar o invocar una API.

Solución

Verifique que OpenSSL utilice TLS 1.1 o una versión posterior para establecer la conexión. Este debería ser el método predeterminado en la mayoría de las distribuciones de Linux o en Windows versión 7 y posteriores. Es posible que los usuarios de macOS deban actualizar OpenSSL.

InvalidRequestException mensaje

InvalidRequestException Recibo cuando intento llamar `CreateDetectorModel` y `UpdateDetectorModel` APIs.

Solución

Para resolver el problema, compruebe lo siguiente. Para obtener más información, consulte [CreateDetectorModel](#) y [UpdateDetectorModel](#).

- Asegúrese de no utilizar al mismo tiempo `seconds` y `durationExpression` como parámetros de `SetTimerAction`.
- Asegúrese de que su expresión de cadena para `durationExpression` sea válida. La expresión de cadena puede contener números, variables (`$variable.<variable-name>`) o valores de entrada (`$input.<input-name>.<path-to-datum>`).

action.setTimerErrores de Amazon CloudWatch Logs

Puede configurar Amazon CloudWatch Logs para supervisar las instancias AWS IoT Events del modelo de detector. Los siguientes son errores comunes que se AWS IoT Events generan cuando se utiliza `action.setTimer`.

- Error: Su expresión de duración para el temporizador llamado `<timer-name>` no se ha podido evaluar como número.

Solución

Asegúrese de que su expresión de cadena para `durationExpression` pueda convertirse en un número. No se permiten otros tipos de datos, como los booleanos.

- Error: El resultado evaluado de su expresión de duración para el temporizador llamado `<timer-name>` es mayor que 31622440. Para garantizar la precisión, asegúrese de que su expresión de duración haga referencia a un valor comprendido entre 60 y 31622400.

Solución

Asegúrese de que la duración de su temporizador sea menor o igual que 31622400 segundos. El resultado evaluado de la duración se redondea hacia abajo al número entero más próximo.

- Error: El resultado evaluado de su expresión de duración para el temporizador llamado `<timer-name>` es menor que 60. Para garantizar la precisión, asegúrese de que su expresión de duración haga referencia a un valor comprendido entre 60 y 31622400.

Solución

Asegúrese de que la duración de su temporizador sea mayor o igual que 60 segundos. El resultado evaluado de la duración se redondea hacia abajo al número entero más próximo.

- Error: Su expresión de duración para el temporizador llamado `<timer-name>` no se ha podido evaluar. Compruebe los nombres de las variables, los nombres de las entradas y las rutas a los datos para asegurarse de que hace referencia a variables y entradas existentes.

Solución

Asegúrese de que su expresión de cadena haga referencia a variables y entradas existentes. La expresión de cadena puede contener números, variables (`$variable.variable-name`) y valores de entrada (`$input.input-name.path-to-datum`).

- Error: No se ha podido establecer el temporizador llamado `<timer-name>`. Compruebe su expresión de duración e inténtelo de nuevo.

Solución

Consulte la [SetTimerAction](#) acción para asegurarse de haber especificado los parámetros correctos y, a continuación, vuelva a configurar el temporizador.

Para obtener más información, consulte [Habilitar el CloudWatch registro de Amazon al desarrollar modelos de AWS IoT Events detectores](#).

Errores de CloudWatch carga útil de Amazon

Puede configurar Amazon CloudWatch Logs para supervisar las instancias AWS IoT Events del modelo de detector. Los siguientes son los errores y advertencias más comunes que se generan al configurar la carga útil de la acción.

- Error: No se ha podido evaluar su expresión para la acción. Asegúrese de que los nombres de las variables, los nombres de las entradas y las rutas a los datos hagan referencia a variables y valores de entrada existentes. Verifique también que el tamaño de la carga sea inferior a 1 KB, el tamaño máximo permitido para una carga.

Solución

Asegúrese de introducir correctamente los nombres de las variables, los nombres de las entradas y las rutas a los datos. También puede recibir este mensaje de error si la carga de acción es superior a 1 KB.

- Error: No se ha podido interpretar su expresión de contenido para la carga de *<action-type>*. Introduzca una expresión de contenido con la sintaxis correcta.

Solución

La expresión de contenido puede contener cadenas ('*string*'), variables (*\$variable.variable-name*), valores de entrada (*\$input.input-name.path-to-datum*), concatenaciones de cadenas y cadenas que contengan *\${}*.

- Error: la expresión de carga útil *{expression}* no es válida. El tipo de carga útil definido es JSON, por lo que debes especificar una expresión que AWS IoT Events dé como resultado una cadena.

Solución

Si el tipo de carga útil especificado es JSON, AWS IoT Events primero comprueba si el servicio puede evaluar la expresión en una cadena. El resultado evaluado no puede ser un booleano ni un número. Podría recibir este error si falla la validación.

- Advertencia: La acción se ejecutó, pero no se ha podido evaluar su expresión de contenido para la carga de acción como JSON válido. El tipo de carga definido es JSON.

Solución

Asegúrese de que AWS IoT Events pueda evaluar su expresión de contenido para la carga útil de la acción en un JSON válido, si define el tipo de carga útil como JSON. AWS IoT Events ejecuta la acción aunque no pueda evaluar la expresión de contenido para convertirla en un JSON válido.

Para obtener más información, consulte [Habilitar el CloudWatch registro de Amazon al desarrollar modelos de AWS IoT Events detectores](#).

Tipos de datos incompatibles

Se han encontrado tipos de datos incompatibles [<inferred-types>] para <reference> en la siguiente expresión: <expression>

Solución

Podría recibir este error por uno de los siguientes motivos:

- Los resultados evaluados de sus referencias no son compatibles con otros operandos de sus expresiones.
- El tipo del argumento pasado a una función no es compatible.

Cuando utilice referencias en expresiones, compruebe lo siguiente:

- Cuando utilice una referencia como operando con uno o más operadores, asegúrese de que todos los tipos de datos a los que hace referencia sean compatibles.

Por ejemplo, en la siguiente expresión, el entero 2 es un operando de los operadores == y &&. Para asegurarse de que los operandos sean compatibles, `$variable.testVariable + 1` y `$variable.testVariable` deben hacer referencia a un número entero o decimal.

Además, el entero 1 es un operando del operador +. Por lo tanto, `$variable.testVariable` debe hacer referencia a un número entero o decimal.

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

- Cuando utilice una referencia como argumento pasado a una función, asegúrese de que la función admita los tipos de datos a los que hace referencia.

Por ejemplo, la siguiente función `timeout("time-name")` requiere una cadena con comillas dobles como argumento. Si utiliza una referencia para el `timer-name` valor, debe hacer referencia a una cadena con comillas dobles.

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

 Note

Para la `convert(type, expression)` función, si utiliza una referencia para el *type* valor, el resultado evaluado de la referencia debe ser `StringDecimal`, o `Boolean`.

Para obtener más información, consulte [AWS IoT Events referencia para las entradas y variables en las expresiones](#).

No se pudo enviar el mensaje a AWS IoT Events

Mensaje: No se ha podido enviar el mensaje a lot Events

Solución

Podría experimentar este error por las siguientes razones:

- La carga del mensaje de entrada no contiene el `Input attribute Key`.
- La `Input attribute Key` no se encuentra en la misma ruta JSON que la especificada en la definición de entrada.
- El mensaje de entrada no coincide con el esquema, tal como se define en la AWS IoT Events entrada.

 Note

La ingesta de datos de otros servicios también experimentará fallos.

Example

Por ejemplo AWS IoT Core, en, la AWS IoT regla fallará con el siguiente mensaje `Verify the Input Attribute key`.

Para resolver este problema, asegúrese de que el esquema del mensaje de carga útil de entrada se ajuste a la definición AWS IoT Events de entrada y que la `Input attribute Key` ubicación coincida. Para obtener más información, consulte [Cree una entrada para los modelos en AWS IoT Events](#) para aprender a definir AWS IoT Events las entradas.

Solución de problemas de un modelo de detector mediante la ejecución de análisis en AWS IoT Events

AWS IoT Events puede analizar su modelo de detector y generar resultados de análisis sin enviar datos de entrada a su modelo de detector. AWS IoT Events realiza una serie de análisis descritos en esta sección para comprobar el modelo de su detector. Esta solución de problemas avanzada también resume la información de diagnóstico, incluyendo nivel de gravedad y ubicación, para que pueda encontrar y solucionar rápidamente los posibles problemas de su modelo de detector. Para obtener más información sobre los tipos de errores de diagnóstico y los mensajes de su modelo de detector, consulte [Análisis del modelo de detector e información de diagnóstico para AWS IoT Events](#).

Puede usar la AWS IoT Events consola, la [API](#), [AWS Command Line Interface \(AWS CLI\)](#) o el [AWS SDK](#) para ver los mensajes de error de diagnóstico derivados del análisis del modelo de su detector.

Note

- Debe corregir todos los errores antes de poder publicar su modelo de detector.
- Le recomendamos que revise las advertencias y tome las medidas necesarias antes de utilizar su modelo de detector en entornos de producción. De lo contrario, el modelo de detector podría no funcionar como se espera.
- Puede tener hasta 10 análisis en el estado RUNNING al mismo tiempo.

Para obtener información sobre cómo analizar su modelo de detector, consulte [Analice un modelo de detector para AWS IoT Events \(consola\)](#) o [Analice un modelo de detector en AWS IoT Events \(AWS CLI\)](#).

Temas

- [Análisis del modelo de detector e información de diagnóstico para AWS IoT Events](#)
- [Analice un modelo de detector para AWS IoT Events \(consola\)](#)
- [Analice un modelo de detector en AWS IoT Events \(AWS CLI\)](#)

Análisis del modelo de detector e información de diagnóstico para AWS IoT Events

Los análisis del modelo de detector recaban la siguiente información de diagnóstico:

- **Nivel:** el nivel de gravedad del resultado del análisis. En función del nivel de gravedad, los resultados de los análisis se clasifican en tres categorías generales:
 - **Información (INFO):** un resultado de información le notifica sobre un campo significativo en su modelo de detector. Este tipo de resultado no suele requerir una acción inmediata.
 - **Advertencia (WARNING):** un resultado de advertencia llama especialmente la atención sobre los campos que podrían causar problemas en su modelo de detector. Le recomendamos que revise las advertencias y tome las medidas necesarias antes de utilizar su modelo de detector en entornos de producción. De lo contrario, el modelo de detector podría no funcionar como se espera.
 - **Error (ERROR):** un resultado de error le notifica que se ha encontrado un problema en su modelo de detector. AWS IoT Events realiza automáticamente este conjunto de análisis al intentar publicar el modelo de detector. Debe corregir todos los errores antes de poder publicar el modelo de detector.
- **Ubicación:** contiene información que puede utilizar para localizar el campo en su modelo de detector al que hace referencia el resultado del análisis. Una ubicación suele incluir el nombre del estado, el nombre del evento de transición, el nombre del evento y la expresión (por ejemplo, `in state TemperatureCheck in onEnter in event Init in action setVariable`).
- **Tipo:** el resultado del tipo de análisis. Los tipos de análisis se dividen en las siguientes categorías:
 - **supported-actions**— AWS IoT Events puede invocar acciones cuando se detecta un evento específico o un evento de transición. Puede definir acciones integradas para utilizar un temporizador o establecer una variable, o enviar datos, a otros servicios de AWS. Debe especificar las acciones que funcionen con otros AWS servicios en una AWS región en la que los AWS servicios estén disponibles.
 - **service-limits**— Las cuotas de servicio, también conocidas como límites, son la cantidad máxima o mínima de recursos u operaciones de servicio para su AWS cuenta. A menos que se indique lo contrario, cada cuota es específica de la región. En función de sus necesidades empresariales, puede actualizar su modelo de detector para evitar encontrarse con límites o solicitar un aumento de cuota. Puede solicitar aumentos para algunas cuotas, pero otras no se pueden aumentar. Para obtener más información, consulte [Cuotas](#).

- **structure:** el modelo de detector debe tener todos los componentes requeridos, como los estados, y seguir una estructura que AWS IoT Events admita. Un modelo de detector debe tener al menos un estado y una condición que evalúe los datos de entrada a fin de detectar eventos significativos. Al detectar un evento, el modelo de detector realiza una transición al siguiente estado y puede invocar acciones. Estos eventos se conocen como eventos de transición. Un evento de transición debe indicar el siguiente estado al que se debe entrar.
- **expression-syntax:** AWS IoT Events proporciona varias formas de especificar valores al crear y actualizar modelos de detectores. Puede utilizar literales, operadores, funciones, referencias y plantillas de sustitución en las expresiones. Puede usar expresiones para especificar valores literales o AWS IoT Events puede evaluar las expresiones antes de especificar valores determinados. La expresión debe seguir la sintaxis requerida. Para obtener más información, consulte [Expresiones para filtrar, transformar y procesar datos de eventos](#).

Las expresiones del modelo de detector AWS IoT Events pueden hacer referencia a datos específicos o a un recurso.

- **data-type:** AWS IoT Events admite tipos de datos enteros, decimales, de cadena y booleanos. Si AWS IoT Events puede convertir automáticamente los datos de un tipo de datos en otro durante la evaluación de la expresión, estos tipos de datos son compatibles.

 Note

- Los únicos tipos de datos compatibles que AWS IoT Events admite son enteros y decimales.
- AWS IoT Events no puede evaluar expresiones aritméticas porque no AWS IoT Events puede convertir un entero en una cadena.

- **referenced-data:** debe definir los datos referenciados en su modelo de detector antes de poder utilizarlos. Por ejemplo, si desea enviar datos a una tabla de DynamoDB, debe definir una variable que haga referencia al nombre de la tabla antes de poder utilizar la variable en una expresión (`$variable.TableName`).
- **referenced-resource:** los recursos que el modelo de detector utilice deben estar disponibles. Debe definir los recursos antes de poder utilizarlos. Por ejemplo, su deseo es crear un modelo de detector para supervisar la temperatura de un invernadero. Debe definir una entrada (`$input.TemperatureInput`) para dirigir los datos de temperatura entrantes a su modelo de detector antes de poder utilizar la `$input.TemperatureInput.sensorData.temperature` para referenciar la temperatura.

Consulte la siguiente sección para solucionar errores y encontrar posibles soluciones a partir del análisis de su modelo de detector.

Solucione los errores del modelo del detector en AWS IoT Events

Los tipos de errores descritos anteriormente proporcionan información de diagnóstico sobre un modelo de detector y corresponden a mensajes que podría recuperar. Utilice estos mensajes y las soluciones sugeridas para resolver los errores de su modelo de detector.

Mensajes y soluciones

- [Location](#)
- [supported-actions](#)
- [service-limits](#)
- [structure](#)
- [expression-syntax](#)
- [data-type](#)
- [referenced-data](#)
- [referenced-resource](#)

Location

Un resultado de un análisis con información sobre Location, corresponde al siguiente mensaje de error:

- Mensaje: Contiene información adicional sobre el resultado del análisis. Este puede ser de información, advertencia o error.

Solución: Podría recibir este mensaje de error si ha especificado una acción que AWS IoT Events actualmente no admite. Para obtener una lista de las acciones admitidas, consulte [Acciones compatibles para recibir datos y activar acciones en AWS IoT Events](#).

supported-actions

Un resultado de un análisis con información sobre supported-actions, corresponde a los siguientes mensajes de error:

- Mensaje: Hay un tipo de acción no válido en la definición de la acción: *action-definition*.

Solución: Podría recibir este mensaje de error si ha especificado una acción que AWS IoT Events actualmente no admite. Para obtener una lista de las acciones admitidas, consulte [Acciones compatibles para recibir datos y activar acciones en AWS IoT Events](#).

- Mensaje: DetectorModel la definición tiene una *aws-service* acción, pero el *aws-service* servicio no es compatible en la región *region-name*.

Solución: es posible que reciba este mensaje de error si la acción que especificó es compatible AWS IoT Events, pero no está disponible en su región actual. Esto puede ocurrir al intentar enviar datos a un AWS servicio que no está disponible en la región. También debes elegir la misma región para ambos AWS IoT Events y los AWS servicios que estás utilizando.

service-limits

Un resultado de un análisis con información sobre service-limits, corresponde a los siguientes mensajes de error:

- Mensaje: La expresión de contenido permitida en la carga útil supera el límite de *content-expression-size* bytes en caso de estar *event-name* en estado *state-name*.

Solución: Podría recibir este mensaje de error si la expresión de contenido de su carga de acción es superior a 1024 bytes. El tamaño de la expresión de contenido de una carga puede ser de hasta 1024 bytes.

- Mensaje: El número de estados permitidos en la definición del modelo de detector ha superado el límite *states-per-detector-model*.

Solución: Podría recibir este mensaje de error si su modelo de detector tiene más de 20 estados. Un modelo de detector puede tener hasta 20 estados.

- Mensaje: La duración del temporizador *timer-name* debe ser de al menos *minimum-timer-duration* segundos.

Solución: Podría recibir este mensaje de error si la duración de su temporizador es inferior a 60 segundos. Se recomienda fijar la duración de un temporizador entre 60 y 31622400 segundos. Si especifica una expresión para la duración de su temporizador, el resultado evaluado de la expresión de duración se redondea al número entero menor más cercano.

- Mensaje: El número de acciones permitidas por evento superó el límite establecido *actions-per-event* en la definición del modelo de detector

Solución: Podría recibir este mensaje de error si el evento tiene más de 10 acciones. Puede tener hasta 10 acciones por cada evento en su modelo de detector.

- Mensaje: El número de eventos de transición permitidos por estado superó el límite establecido *transition-events-per-state* en la definición del modelo de detector.

Solución: Podría recibir este mensaje de error si el estado tiene más de 20 eventos de transición. Puede tener hasta 20 eventos de transición por cada estado en su modelo de detector.

- Mensaje: El número de eventos permitidos por estado superó el límite *events-per-state* en la definición del modelo de detector

Solución: Podría recibir este mensaje de error si el estado tiene más de 20 eventos. Puede tener hasta 20 eventos por cada estado en su modelo de detector.

- Mensaje: El número máximo de modelos de detectores que pueden asociarse a una única entrada podría haber alcanzado el límite. *input-name* La entrada se utiliza en las rutas del modelo de *detector-models-per-input* detector.

Solución: Podría recibir este mensaje de advertencia si intentara dirigir una entrada a más de 10 modelos de detectores. Puede tener hasta 10 modelos de detectores diferentes asociados a un único modelo de detector.

structure

Un resultado de un análisis con información sobre `structure`, corresponde a los siguientes mensajes de error:

- Mensaje: Es posible que las acciones solo tengan un tipo definido, pero se encontró una acción con *number-of-types* tipos. Divídala en acciones separadas.

Solución: Podría recibir este mensaje de error si ha especificado dos o más acciones en un único campo al utilizar operaciones de la API para crear o actualizar su modelo de detector. Puede definir una matriz de objetos de `Action`. Asegúrese de definir cada acción como un objeto independiente.

- Mensaje: Las `TransitionEvent` *transition-event-name* transiciones a un estado *state-name* inexistente.

Solución: es posible que reciba este mensaje de error si AWS IoT Events no encuentra el siguiente estado al que hacía referencia el evento de transición. Asegúrese de que el siguiente estado esté definido y de haber introducido el nombre de estado correcto.

- Mensaje: DetectorModelDefinition Tenían un nombre de estado compartido: se encontró un estado *state-name* con *number-of-states* repeticiones.

Solución: Podría recibir este mensaje de error si utiliza el mismo nombre para uno o más estados. Asegúrese de asignar un nombre único a cada estado en su modelo de detector. El nombre del estado debe tener entre 1 y 128 caracteres. Caracteres válidos: a-z, A-Z, 0-9, _ (guion bajo) y - (guion).

- Mensaje: Las definiciones initialStateName *initial-state-name* no correspondían a un estado definido.

Solución: Podría recibir este mensaje de error si el nombre del estado inicial es incorrecto. El modelo de detector permanece en el estado inicial (inicio) hasta que llegue una entrada. En cuanto llega una entrada, el modelo de detector pasa de inmediato al siguiente estado. Asegúrese de que el nombre del estado inicial sea el nombre de un estado definido y de haber introducido el nombre correcto.

- Mensaje: La definición del modelo de detector debe utilizar al menos una entrada en una condición.

Solución: Podría recibir este error si no ha especificado una entrada en una condición. Debe utilizar al menos una entrada en al menos una condición. De lo contrario, AWS IoT Events no evalúa los datos entrantes.

- Mensaje: solo se puede configurar uno de los segundos y DurationExpression. SetTimer

Solución: Podría recibir este mensaje de error si utiliza tanto seconds como durationExpression para su temporizador. Asegúrese de utilizar seconds o durationExpression como parámetros de SetTimerAction. Para obtener más información, consulta [SetTimerAction](#) en la AWS IoT Events Referencia de la API de .

- Mensaje: Una acción de su modelo de detector es inaccesible. Compruebe la condición que inicia la acción.

Solución: Si una acción en su modelo de detector es inaccesible, la condición del evento da como resultado falso. Compruebe la condición del evento que contiene la acción a fin de asegurarse

de que dé como resultado verdadero. Si la condición del evento da como resultado verdadero, la acción debería estar disponible.

- Mensaje: Se está leyendo un atributo de entrada, pero esto podría deberse al vencimiento de un temporizador.

Solución: El valor de un atributo de entrada se puede leer cuando ocurre una de las siguientes situaciones:

- Se ha recibido un nuevo valor de entrada.
- Cuando un temporizador del detector ha vencido.

Para asegurarse de que un atributo de entrada se evalúe solo al recibirse un nuevo valor para esa entrada, incluya una llamada a la función `triggerType("Message")` en su condición de la siguiente manera:

La condición original que se evalúa en el modelo de detector:

```
if ($input.HeartBeat.status == "OFFLINE")
```

pasaría a ser similar a la siguiente:

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

donde la llamada a la función `triggerType("Message")` viene antes de la entrada inicial proporcionada en la condición. Mediante esta técnica, la función `triggerType("Message")` se evaluará como verdadero y satisfará la condición de recibir un nuevo valor de entrada. Para obtener más información sobre el uso de la función `triggerType`, consulte [triggerType](#) en la sección [Expresiones](#) de la Guía para desarrolladores de AWS IoT Events

- Mensaje: Un estado de su modelo de detector es inaccesible. Compruebe la condición que provocará una transición al estado deseado.

Solución: Si un estado en su modelo de detector es inaccesible, una condición que cause una transición entrante a ese estado se evaluará como falsa. Compruebe que las condiciones de las transiciones entrantes a ese estado inaccesible en su modelo de detector den como resultado verdadero, para que el estado deseado se vuelva accesible.

- Mensaje: Un temporizador que vence puede provocar el envío de una cantidad inesperada de mensajes.

Solución: Para evitar que su modelo de detector entre en un estado infinito de envío de una cantidad inesperada de mensajes porque un temporizador ha vencido, considere el uso de una llamada a la función `triggerType("Message")` en las condiciones de su modelo de detector de la siguiente manera:

La condición original que se evalúa en el modelo de detector:

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

se transformaría en una condición similar a la siguiente:

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

donde la llamada a la función `triggerType("Message")` viene antes de la entrada inicial proporcionada en la condición.

Este cambio evita que se inicien acciones de temporizador en su detector, lo que impide un bucle infinito de envío de mensajes. Para obtener más información sobre cómo utilizar las acciones de temporizador en su detector, consulte la página [Uso de las acciones integradas](#) de la Guía para desarrolladores de AWS IoT Events

expression-syntax

Un resultado de un análisis con información sobre `expression-syntax`, corresponde a los siguientes mensajes de error:

- Mensaje: La expresión de carga útil `{expression}` no es válida. El tipo de carga útil definido es JSON, por lo que debes especificar una expresión que AWS IoT Events dé como resultado una cadena.

Solución: si el tipo de carga útil especificado es JSON, AWS IoT Events primero comprueba si el servicio puede evaluar la expresión en una cadena. El resultado evaluado no puede ser un booleano ni un número. Si la validación no tiene éxito, podría recibir este error.

- Mensaje: `SetVariableAction.value` debe ser una expresión. No se pudo analizar el valor "`variable-value`"

Solución: Puede utilizar `SetVariableAction` para definir una variable con un `name` y un `value`. El `value` puede ser una cadena, un número o un valor booleano. También puede especificar una expresión para el `value`. Para obtener más información [SetVariableAction](#), consulte la referencia de la AWS IoT Events API.

- Mensaje: No hemos podido analizar su expresión de los atributos (*attribute-name*) para la acción de DynamoDB. Introduzca la expresión con la sintaxis correcta.

Solución: Debe utilizar expresiones para todos los parámetros en las plantillas de sustitución de `DynamoDBAction`. Para obtener más información, consulte [Dynamo DBAction](#) en la referencia de la AWS IoT Events API.

- Mensaje: No hemos podido analizar tu expresión del `TableName` para la acción `Dynamo. DBv2`. Introduzca la expresión con la sintaxis correcta.

Solución: El `tableName` en `DynamoDBv2Action` debe ser una cadena. Debe utilizar una expresión para el `tableName`. Las expresiones aceptan literales, operadores, funciones, referencias y plantillas de sustitución. Para obtener más información, consulte [Dynamo DBv2 Action](#) en la referencia de la API.AWS IoT Events

- Mensaje: No se ha podido evaluar su expresión como JSON válido. La `DBv2` acción `Dynamo` solo admite el tipo de carga útil JSON.

Solución: El tipo de carga para `DynamoDBv2` debe ser JSON. Asegúrese de AWS IoT Events poder evaluar su expresión de contenido para la carga útil en un JSON válido. Para obtener más información, consulta [Dynamo DBv2 Action](#), en la referencia de la AWS IoT Events API.

- Mensaje: No hemos podido analizar tu expresión de contenido para ver la carga útil de. *action-type* Introduzca una expresión de contenido con la sintaxis correcta.

Solución: la expresión de contenido puede contener cadenas ('*string*') y variables (`$variable`). *variable-name*), valores de entrada (`$input.input-name.path-to-datum`), las concatenaciones de cadenas y las cadenas que contienen. `${}`

- Mensaje: Las cargas personalizadas no deben estar vacías.

Solución: es posible que recibas este mensaje de error si has elegido una carga útil personalizada para tu acción y no has introducido ninguna expresión de contenido en la consola. AWS IoT Events Si elige Carga personalizada, debe introducir una expresión de contenido en Carga personalizada. Para obtener más información consulte [Carga](#) en la Referencia de la API de AWS IoT Events .

- Mensaje: No se pudo analizar la expresión de duración '*duration-expression*' para el temporizador '*timer-name*'.

Solución: El resultado evaluado de su expresión de duración para el temporizador debe ser un valor comprendido entre 60 y 31622400. El resultado evaluado de la duración se redondea hacia abajo al número entero más próximo.

- Mensaje: No se pudo analizar la expresión '*expression*' para *action-name*

Solución: Podría recibir este mensaje si la expresión para la acción especificada tiene una sintaxis incorrecta. Asegúrese de introducir una expresión con la sintaxis correcta. Para obtener más información, consulte [Sintaxis para filtrar los datos del dispositivo y definir acciones en AWS IoT Events](#).

- Mensaje: No se ha podido analizar tu *fieldName* formulario. Debe utilizar la sintaxis correcta en su expresión.

Solución: es posible que recibas este error si no has podido analizar tu formulario *fieldName*. Asegúrese de que *fieldName* utiliza una expresión que AWS IoT Events pueda analizarse. Para obtener más información, consulta [IoT SiteWise Action](#) en la AWS IoT Events Referencia de la API de .

data-type

Un resultado de un análisis con información sobre data-type, corresponde a los siguientes mensajes de error:

- Mensaje: La expresión *duration-expression* de duración del temporizador no *timer-name* es válida, debe devolver un número.

Solución: es posible que reciba este mensaje de error AWS IoT Events si no puede evaluar la expresión de duración del temporizador con un número. Asegúrese de que su *durationExpression* se pueda convertir en un número. Otros tipos de datos, como los booleanos, no son compatibles.

- Mensaje: *condition-expression* La expresión no es una expresión condicionada válida.

Solución: es posible que reciba este mensaje de error si AWS IoT Events no puede evaluarla *condition-expression* con un valor booleano. El valor booleano debe ser TRUE o FALSE. Asegúrese de que su expresión de condición se pueda convertir en un valor booleano. Si el

resultado no es un valor booleano, es equivalente a FALSE y no invocará las acciones o la transición al nextState especificado en el evento.

- Mensaje: Se han encontrado tipos de datos incompatibles [*inferred-types*] *reference* en la siguiente expresión: *expression*

Solución:: Todas las expresiones para el mismo atributo de entrada o variable en el modelo de detector deben hacer referencia al mismo tipo de datos.

Utilice la siguiente información para resolver el problema:

- Cuando utilice una referencia como operando con uno o más operadores, asegúrese de que todos los tipos de datos a los que hace referencia sean compatibles.

Por ejemplo, en la siguiente expresión, el entero 2 es un operando de los operadores == y &&. Para asegurarse de que los operandos sean compatibles, `$variable.testVariable + 1` y `$variable.testVariable` deben hacer referencia a un número entero o decimal.

Además, el entero 1 es un operando del operador +. Por lo tanto, `$variable.testVariable` debe hacer referencia a un número entero o decimal.

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

- Cuando utilice una referencia como argumento pasado a una función, asegúrese de que la función admita los tipos de datos a los que hace referencia.

Por ejemplo, la siguiente función `timeout("time-name")` requiere una cadena con comillas dobles como argumento. Si utiliza una referencia para el *timer-name* valor, debe hacer referencia a una cadena con comillas dobles.

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

Note

Para la `convert(type, expression)` función, si utiliza una referencia para el *type* valor, el resultado evaluado de la referencia debe ser `StringDecimal`, o `Boolean`.

Para obtener más información, consulte [AWS IoT Events referencia para las entradas y variables en las expresiones](#).

- Mensaje: Se utilizan tipos de datos incompatibles [*inferred-types*] con *reference*. Esto podría provocar un error en el tiempo de ejecución.

Solución: Podría recibir este mensaje de advertencia si dos expresiones para el mismo atributo de entrada o variable hacen referencia a dos tipos de datos. Asegúrese de que sus expresiones para el mismo atributo de entrada o variable hagan referencia al mismo tipo de datos en el modelo de detector.

- Mensaje: Los tipos de datos [*inferred-types*] que ha introducido para el operador [*operator*] no son compatibles con la siguiente expresión: '*expression*'

Solución: Podría recibir este mensaje de error si su expresión combina tipos de datos que no son compatibles con un operador especificado. Por ejemplo, en la siguiente expresión, el operador + es compatible con los tipos de datos entero, decimal y cadena, pero no con los operandos de tipo de datos booleano.

```
true + false
```

Debe asegurarse de que los tipos de datos que utilice con un operador sean compatibles.

- Mensaje: Los tipos de datos [*inferred-types*] encontrados para *input-attribute* no son compatibles y pueden provocar un error de tiempo de ejecución.

Solución: Podría recibir este mensaje de error si dos expresiones para el mismo atributo de entrada hacen referencia a dos tipos de datos ya sea el `OnEnterLifecycle` de un estado, o tanto el `OnInputLifecycle` como el `OnExitLifecycle` de un estado. Asegúrese de que sus expresiones en `OnEnterLifecycle` (o tanto en `OnInputLifecycle` como en `OnExitLifecycle`) hagan referencia al mismo tipo de datos para cada estado de su modelo de detector.

- Mensaje: La expresión de carga útil [*expression*] no es válida. Especifique una expresión que se evaluaría como cadena en tiempo de ejecución porque el tipo de carga útil es en formato JSON.

Solución: es posible que recibas este error si el tipo de carga útil especificado es JSON, pero no AWS IoT Events puedes evaluar su expresión en una cadena. Asegúrese de que el resultado evaluado sea una cadena, no un booleano ni un número.

- Mensaje: La expresión interpolada {*interpolated-expression*} debe dar como resultado un valor entero o booleano en tiempo de ejecución. De lo contrario, la expresión de carga útil {*payload-expression*} no se podrá analizar en tiempo de ejecución como un JSON válido.

Solución: es posible que recibas este mensaje de error si no AWS IoT Events puedes evaluar la expresión interpolada en un valor entero o booleano. Asegúrese de que su expresión interpolada puede convertirse a un valor entero o booleano, dado que otros tipos de datos, como cadena, no son compatibles.

- Mensaje: el tipo de expresión del `IotSitusewiseAction` campo *expression* se define como tipo *defined-type* y se deduce como tipo. *inferred-type* El tipo definido y el tipo inferido deben ser iguales.

Solución: Podría recibir este mensaje de error si su expresión en el `propertyValue` de `IotSitusewiseAction` tiene un tipo de datos definido diferente al tipo de datos inferido por AWS IoT Events. Asegúrese de utilizar el mismo tipo de datos para todas las instancias de esta expresión en su modelo de detector.

- Mensaje: Los tipos de datos [*inferred-types*] utilizados para la `setTimer` acción no se evalúan en la siguiente expresión: Integer *expression*

Solución: Podría recibir este mensaje de error si el tipo de datos inferido para su expresión de duración no es un valor entero o decimal. Asegúrese de que su `durationExpression` se pueda convertir a un número. Otros tipos de datos, como los booleanos y de cadena, no son compatibles.

- Mensaje: Los tipos de datos [*inferred-types*] utilizados con los operandos del operador de comparación [*operator*] no son compatibles en la siguiente expresión: *expression*

Solución: los tipos de datos inferidos para los operandos de *operator* la expresión condicional (*expression*) del modelo de detector no coinciden. Los operandos se deben utilizar con tipos de datos coincidentes en todas las demás partes de su modelo de detector.

Tip

Puede utilizar `convert` para cambiar el tipo de datos de una expresión en su modelo de detector. Para obtener más información, consulte [Funciones para usar en expresiones AWS IoT Events](#).

referenced-data

Un resultado de un análisis con información sobre `referenced-data`, corresponde a los siguientes mensajes de error:

- Mensaje: Se ha detectado una avería del temporizador: el temporizador *timer-name* se utiliza en una expresión, pero nunca se configura.

Solución: Podría recibir este mensaje de error si utiliza un temporizador que no se haya configurado. Debe configurar cualquier temporizador antes de utilizarlo en una expresión. Asegúrese también de introducir el nombre correcto del temporizador.

- Mensaje: Variable rota detectada: la variable *variable-name* se usa en una expresión pero nunca se establece.

Solución: Podría recibir este mensaje de error si utiliza una variable que no se haya configurado. Debe configurar cualquier variable antes de utilizarla en una expresión. Asegúrese también de introducir el nombre correcto de la variable.

- Mensaje: Se ha detectado una variable inservible: se utiliza una variable en una expresión antes de haberle establecido un valor.

Solución: A cada variable se le debe asignar un valor antes de que se pueda evaluar en una expresión. Establezca el valor de la variable antes de cada uso para poder así recuperarlo. Asegúrese también de introducir el nombre correcto de la variable.

referenced-resource

Un resultado de un análisis con información sobre `referenced-resource`, corresponde a los siguientes mensajes de error:

- Mensaje: La definición del modelo de detector contiene una referencia a una entrada que no existe.

Solución: Podría recibir este mensaje de error si utiliza expresiones para hacer referencia a una entrada que no existe. Asegúrese de que su expresión haga referencia a una entrada existente e introduzca el nombre de entrada correcto. Si no tiene una entrada, primero cree una.

- Mensaje: La definición del modelo de detector no contiene lo InputName siguiente: *input-name*

Solución: Podría recibir este mensaje de error si su modelo de detector contiene un nombre de entrada no válido. Asegúrese de introducir el nombre de la entrada correcto. El nombre de la entrada debe tener entre 1 y 128 caracteres. Caracteres válidos: a-z, A-Z, 0-9, _ (guion bajo) y - (guion).

Analice un modelo de detector para AWS IoT Events (consola)

AWS IoT Events le permite monitorear y reaccionar a los datos de IoT mediante la detección de eventos y la activación de acciones con la AWS IoT Events API. En los siguientes pasos, se utiliza la AWS IoT Events consola para analizar un modelo de detector.

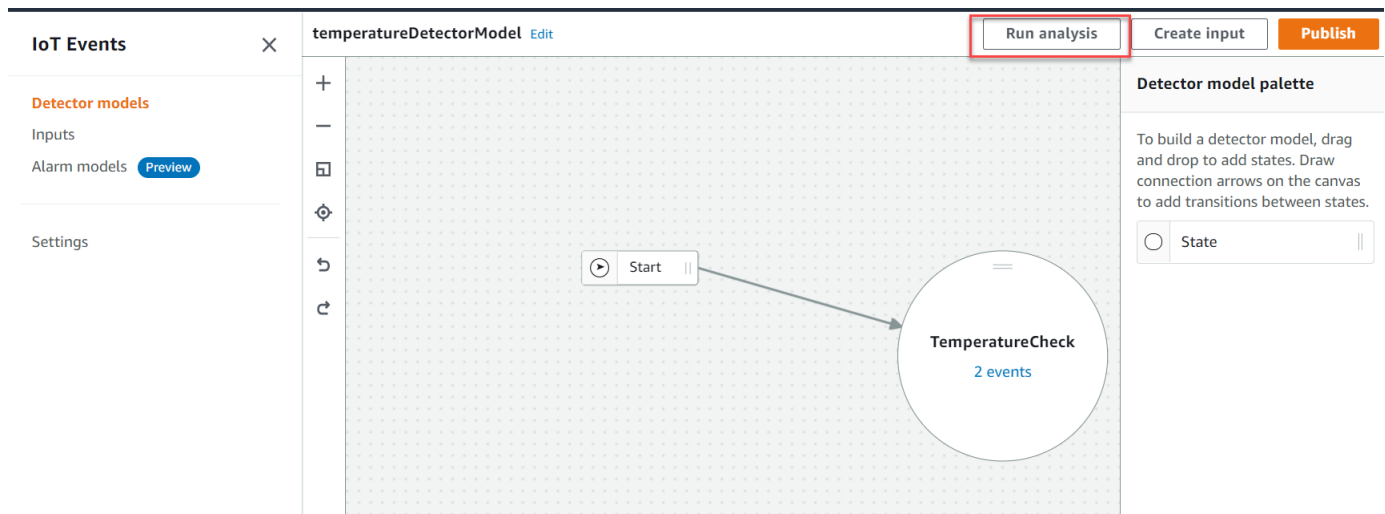
Note

Una vez que AWS IoT Events comience a analizar el modelo de detector, dispondrá de hasta 24 horas para recuperar los resultados del análisis.

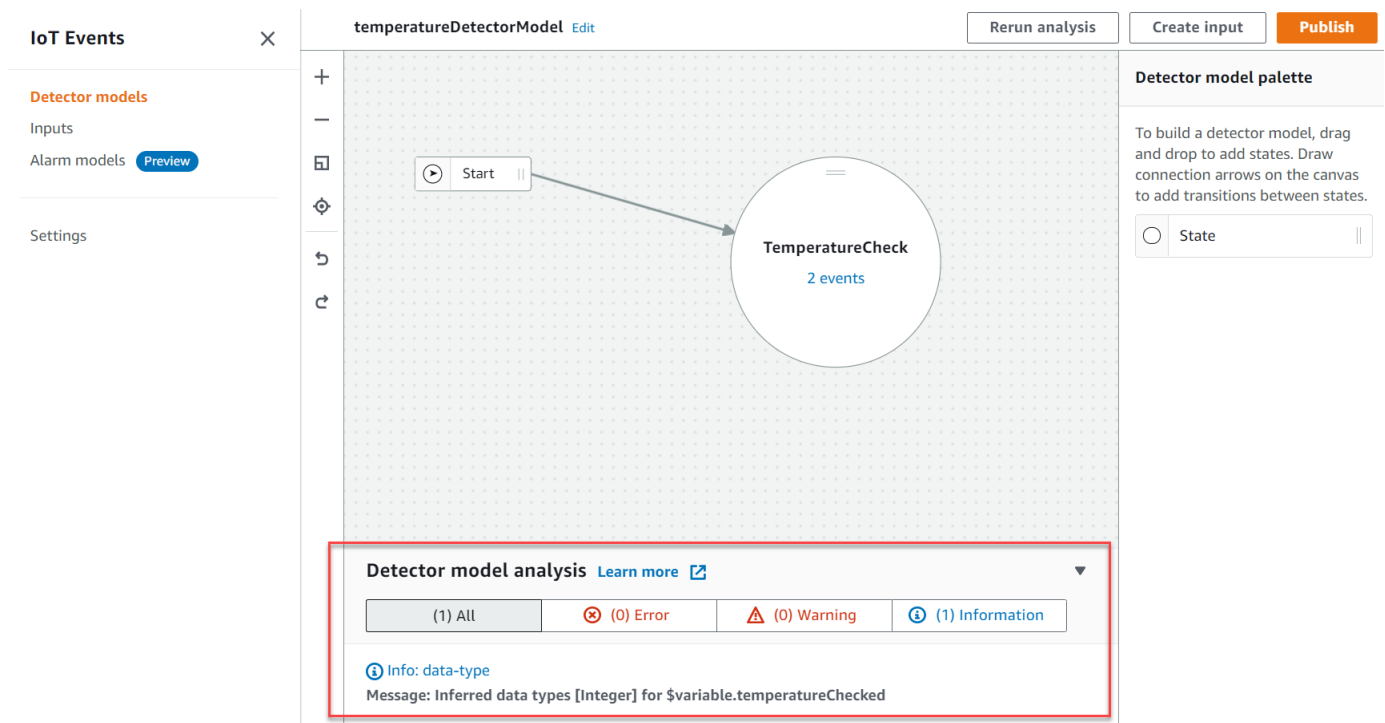
El análisis del modelo de un detector puede ayudarle a optimizar sus modelos, identificar posibles problemas y garantizar que funcionan según lo previsto. Por ejemplo, en un parque eólico, el análisis del modelo del detector podría revelar si el modelo identifica correctamente las posibles fallas de los engranajes en función de patrones de vibración anormales. O bien, si el modelo activa con precisión las alertas de mantenimiento cuando la velocidad del viento supera los umbrales de operación seguros. Al refinar un modelo basado en el análisis, puede mejorar el mantenimiento predictivo, reducir el tiempo de inactividad y mejorar la eficiencia general de la producción de energía.

Para analizar un modelo de detector

1. Inicie sesión en la [consola de AWS IoT Events](#).
2. En el panel de navegación, seleccione Modelos de detectores.
3. En Modelos de detectores, seleccione el modelo de detector deseado.
4. En la página del modelo de detector, seleccione Editar.
5. En la esquina superior derecha, seleccione Ejecutar análisis.



El siguiente es un ejemplo del resultado de un análisis en la AWS IoT Events consola.



Analice un modelo de detector en AWS IoT Events (AWS CLI)

El análisis programático de los modelos de sus AWS IoT Events detectores proporciona información valiosa sobre su estructura, comportamiento y rendimiento. Este enfoque basado en la API permite el análisis automatizado, la integración con los flujos de trabajo existentes y la capacidad de realizar operaciones masivas en varios modelos de detectores. Al aprovechar la [StartDetectorModelAnalysis](#) API, puede iniciar exámenes exhaustivos de sus modelos, lo

que le ayudará a identificar posibles problemas, optimizar los flujos lógicos y garantizar que su procesamiento de eventos de IoT se alinee con los requisitos de su negocio.

Los siguientes pasos se utilizan AWS CLI para analizar un modelo de detector.

Para analizar un modelo de detector mediante AWS CLI

1. Ejecute el siguiente comando para iniciar un análisis.

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

 Note

file-name Sustitúyalo por el nombre del archivo que contiene la definición del modelo de detector.

Example Definición del modelo de detector

```
{
  "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"
}

```

Si utiliza AWS CLI para analizar un modelo de detector existente, elija una de las siguientes opciones para recuperar la definición del modelo de detector:

- Si desea utilizar la AWS IoT Events consola, haga lo siguiente:
 1. En el panel de navegación, seleccione Modelos de detectores.
 2. En Modelos de detectores, seleccione el modelo de detector deseado.
 3. Seleccione Exportar modelo de detector en Acción para descargar el modelo de detector. El modelo de detector se guarda en JSON.
 4. Abra el archivo JSON del modelo de detector.
 5. Solo necesita el objeto `detectorModelDefinition`. Elimine lo siguiente:
 - La primera llave (`{`) en la parte superior de la página
 - La línea `detectorModel`
 - El objeto `detectorModelConfiguration`.
 - La última llave (`}`) en la parte inferior de la página

6. Guarde el archivo.

- Si desea utilizar el AWS CLI, haga lo siguiente:

1. Ejecute el comando siguiente en un terminal.

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

2. *detector-model-name* Sustitúyalo por el nombre del modelo de su detector.

3. Copie el objeto `detectorModelDefinition` en un editor de texto.

4. Añada llaves (`{}`) fuera de `detectorModelDefinition`.

5. Guarde el archivo en JSON.

Example Ejemplo de respuesta

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

2. Copie el ID de análisis de la salida.
3. Ejecute el siguiente comando para recuperar el estado del análisis.

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

 Note

analysis-id Sustitúyalo por el identificador de análisis que copió.

Example Ejemplo de respuesta

```
{
  "status": "COMPLETE"
}
```

El estado puede ser uno de los siguientes valores:

- **RUNNING**— AWS IoT Events está analizando su modelo de detector. El proceso puede tardar hasta un minuto en completarse.

- COMPLETE— AWS IoT Events ha terminado de analizar su modelo de detector.
 - FAILED— no AWS IoT Events pudo analizar su modelo de detector. Inténtelo de nuevo más tarde.
4. Ejecute el siguiente comando para recuperar uno o más resultados de análisis del modelo de detector.

 Note

analysis-id Sustitúyalo por el identificador de análisis que copió.

```
aws iotevents get-detector-model-analysis-results --analysis-id "analysis-id"
```

Example Ejemplo de respuesta

```
{
  "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

Una vez que AWS IoT Events comience a analizar el modelo de detector, dispondrá de hasta 24 horas para recuperar los resultados del análisis.

AWS IoT Events comandos

Este capítulo proporciona una guía completa de todas las operaciones de la API disponibles en AWS IoT Events. Ofrece explicaciones detalladas, incluidos ejemplos de solicitudes, respuestas y posibles errores para cada operación en los protocolos de servicios web compatibles. Comprender estas operaciones de API le ayuda a AWS IoT Events integrarse eficazmente en sus aplicaciones de IoT y a automatizar sus flujos de trabajo de detección y respuesta a eventos.

AWS IoT Events acciones

Puede usar los comandos de la AWS IoT Events API para crear, leer, actualizar y eliminar entradas y modelos de detectores, y para enumerar sus versiones. Para obtener más información, consulta las [acciones](#) y [los tipos de datos](#) que se admiten AWS IoT Events en la Referencia de la AWS IoT Events API.

Las [AWS IoT Events secciones](#) de la Referencia de AWS CLI comandos incluyen los AWS CLI comandos que puede usar para administrar y manipular AWS IoT Events.

AWS IoT Events datos

Puede usar los comandos de la API de AWS IoT Events datos para enviar entradas a los detectores, hacer una lista de los detectores y ver o actualizar el estado de un detector. Para obtener más información, consulta las [acciones](#) y [los tipos de datos](#) que admite la referencia de AWS IoT Events Data in the AWS IoT Events API.

[Las secciones de AWS IoT Events datos](#) de la Referencia de AWS CLI comandos incluyen los AWS CLI comandos que puede usar para procesar AWS IoT Events datos.

Historial de documentos para AWS IoT Events

En la siguiente tabla se describen los cambios importantes efectuados en la Guía para desarrolladores de AWS IoT Events después del 17 de septiembre de 2020. Para obtener más información sobre las actualizaciones de esta documentación, puede suscribirse a una fuente RSS.

Cambio	Descripción	Fecha
Notificación del fin del soporte	Aviso de fin del soporte: el 20 de mayo de 2026, AWS suspenderemos el soporte para AWS IoT Events. Después del 20 de mayo de 2026, ya no podrás acceder a la AWS IoT Events consola ni a AWS IoT Events los recursos.	20 de mayo de 2025
Lanzamiento regional	AWS IoT Events ya está disponible en la región de Asia Pacífico (Bombay).	30 de septiembre de 2021
Lanzamiento regional	AWS IoT Events ya está disponible en la región AWS GovCloud (EE. UU.-Oeste).	22 de septiembre de 2021
Solución de problemas de un modelo de detector mediante la ejecución de análisis	AWS IoT Events ahora puede analizar su modelo de detector y generar resultados de análisis que puede utilizar para solucionar los problemas de su modelo de detector.	23 de febrero de 2021
Lanzamiento regional	Lanzado AWS IoT Events en China (Pekín).	30 de septiembre de 2020

[Uso de expresiones](#)

Se han añadido ejemplos para mostrarle cómo se escriben expresiones.

22 de septiembre de 2020

[Monitoreo con alarmas](#)

Las alarmas le ayudan a monitorear sus datos a fin de detectar cambios. Puede crear alarmas que le envíen notificaciones al superarse un umbral.

1 de junio de 2020

Actualizaciones anteriores

En la siguiente tabla se describen los cambios importantes efectuados en la Guía para desarrolladores de AWS IoT Events antes del 18 de septiembre de 2020.

Cambio	Descripción	Fecha
<u>Se agregó la validación de tipos a la referencia de expresiones</u>	Se agregó información de validación de tipos a la referencia de expresiones.	3 de agosto de 2020
<u>Se agregó una advertencia de región para otros servicios</u>	Se agregó una advertencia sobre la selección de la misma región AWS IoT Events y otros AWS servicios.	7 de mayo de 2020
Adiciones, actualizaciones	• Función de personalización de la carga útil • Nuevas acciones de eventos: Amazon DynamoDB y AWS IoT SiteWise	27 de abril de 2020
Se agregaron funciones integradas para las expresiones	Se agregaron funciones integradas para las expresiones	10 de septiembre de 2019

Cambio	Descripción	Fecha
es condicionales del modelo de detector	es condicionales del modelo de detector.	
Se agregaron ejemplos de modelos de detectores	Se agregaron ejemplos para el modelo de detector.	5 de agosto de 2019
Se agregaron nuevas acciones de evento	Se han añadido nuevas acciones de evento para: • Lambda • Amazon SQS • Kinesis Data Firehose • AWS IoT Events entrada	19 de julio de 2019
Adiciones, correcciones	• Descripción actualizada de la <code>timeout()</code> función. • Se ha añadido la práctica recomendada en relación con la inactividad de las cuentas.	11 de junio de 2019
Política de permisos actualizada y opciones de depuración de la consola	• Se actualizó la política de permisos de la consola. • Imagen actualizada de la página de opciones de depuración de la consola.	5 de junio de 2019
Actualizaciones	AWS IoT Events servicio abierto a disponibilidad general.	30 de mayo de 2019

Cambio	Descripción	Fecha
Adiciones, actualizaciones	• Información de seguridad actualizada. • Se agregó un ejemplo de modelo de detector anotado.	22 de mayo de 2019
Se agregaron ejemplos y permisos necesarios	Se agregaron ejemplos de carga útil de Amazon SNS; se agregaron los permisos necesarios para. <code>CreateDetectorModel</code>	17 de mayo de 2019
Se agregó información de seguridad adicional	Se agregó información a la sección de seguridad.	9 de mayo de 2019
Versión inicial de vista previa limitada	Versión inicial de la documentación de vista previa limitada.	28 de marzo de 2019