aws

개발자 가이드

AWS IoT Events

AWS IoT Events: 개발자 가이드

Table of Contents

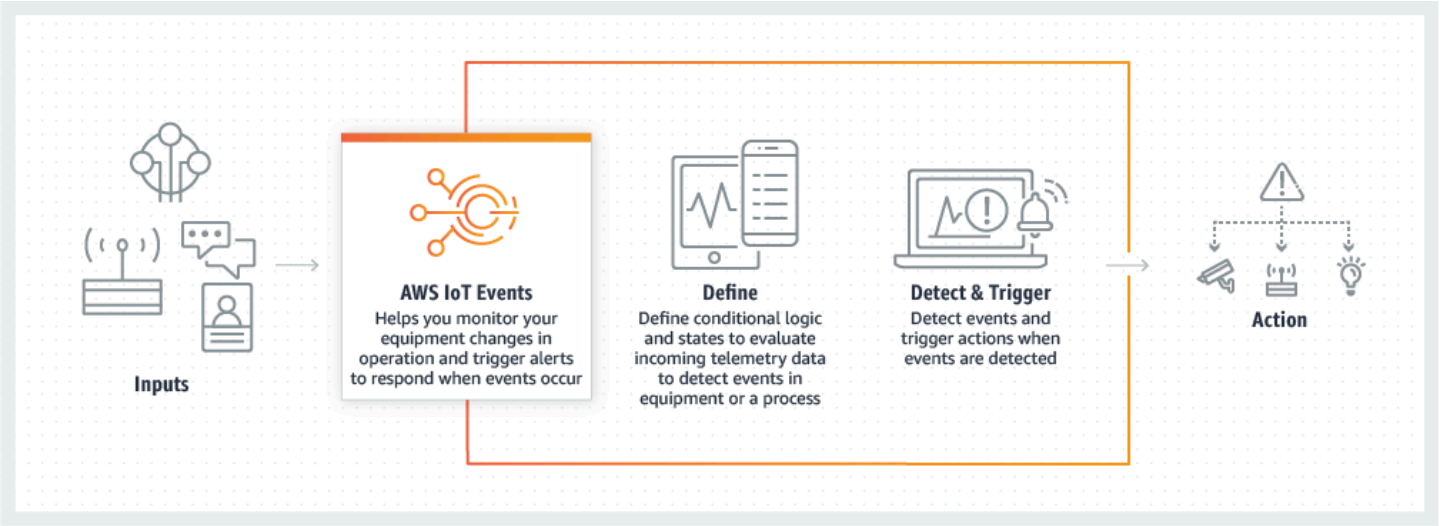
지원 종료 알림: 2026년 5월 20일에 AWS 에 대한 지원이 종료됩니다 AWS IoT Events. 2026년 5월 20일 이후에는 AWS IoT Events 콘솔 또는 AWS IoT Events 리소스에 더 이상 액세스할 수 없습니다. 자세한 내용은 AWS IoT Events 지원 종료를 참조하세요.

기계 번역으로 제공되는 번역입니다. 제공된 번역과 원본 영어의 내용이 상충하는 경우에는 영어 버전이 우선합니다.

AWS IoT Events란 무엇인가요?

AWS IoT Events 를 사용하면 장비 또는 디바이스 플릿에서 장애 또는 운영 변경을 모니터링하고 이러한 이벤트가 발생할 때 작업을 트리거할 수 있습니다.는 디바이스, 프로세스, 애플리케이션 및 기타 AWS 서비스의 IoT 센서 데이터를 AWS IoT Events 지속적으로 모니터링하여 중요한 이벤트를 식별하므로 조치를 취할 수 있습니다.

AWS IoT Events 를 사용하여 AWS IoT Events 콘솔 또는 API를 통해 액세스할 수 있는 AWS 클라우드의 복잡한 이벤트 모니터링 애플리케이션을 빌드합니다. APIs



주제

이점 및 특징

여러 소스의 입력 수락

AWS IoT Events 는 많은 IoT 원격 측정 데이터 소스의 입력을 허용합니다. 여기에는 센서 디바이스, 관리 애플리케이션 및 AWS IoT Core 및와 같은 기타 AWS IoT 서비스가 포함됩니다 AWS IoT Analytics. 표준 API 인터페이스(`BatchPutMessage` API) 또는 AWS IoT Events 콘솔을 사용하여 원격 측정 데이터 입력을 AWS IoT Events 에 푸시할 수 있습니다.

시작하기에 대한 자세한 내용은 섹션을 AWS IoT Events참조하세요AWS IoT Events 콘솔 시작하기.

간단한 논리적 표현식을 사용하여 복잡한 이벤트 패턴 인식

AWS IoT Events 는 단일 IoT 디바이스 또는 애플리케이션 또는 다양한 장비 및 많은 독립 센서의 여러 입력과 관련된 이벤트 패턴을 인식할 수 있습니다. 각 센서와 애플리케이션이 중요한 정보를 제공하기 때문에 이는 특히 유용합니다. 그러나 다양한 센서와 애플리케이션 데이터를 결합해야만 운영 성능과 품질을 완벽하게 파악할 수 있습니다. 복잡한 코드 대신 간단한 논리적 표현식을 사용하여 이러한 이벤트를 인식하도록 AWS IoT Events 감지기를 구성할 수 있습니다.

논리적 표현식에 대한 자세한 내용은 섹션을 참조하세요이벤트 데이터를 필터링, 변환 및 처리하는 표현식.

이벤트를 기반으로 작업 트리거

AWS IoT Events 를 사용하면 Amazon Simple Notification Service(Amazon SNS), AWS IoT Core Lambda, Amazon SQS 및 Amazon Kinesis Firehose에서 작업을 직접 트리거할 수 있습니다. Amazon Connect 또는 자체 엔터프라이즈 리소스 계획(ERP) 애플리케이션과 같은 다른 서비스를 사용하여 작업을 수행할 수 있는 AWS IoT 규칙 엔진을 사용하여 AWS Lambda 함수를 트리거할 수도 있습니다.

AWS IoT Events 에는 수행할 수 있는 사전 구축된 작업 라이브러리가 포함되어 있으며, 이를 통해 자체 작업을 정의할 수도 있습니다.

이벤트를 기반으로 작업을 트리거하는 방법에 대한 자세한 내용은 섹션을 참조하세요에서 데이터를 수신하고 작업을 트리거하는 데 지원되는 작업 AWS IoT Events.

플릿의 요구 사항에 맞게 자동으로 확장

AWS IoT Events 동종 디바이스를 연결할 때가 자동으로 확장됩니다. 특정 유형의 디바이스에 대해 감지기를 한 번 정의할 수 있으며, 서비스는 연결된 해당 디바이스의 모든 인스턴스를 자동으로 확장하고 관리합니다 AWS IoT Events.

감지기 모델의 예를 살펴보려면 섹션을 참조하세요AWS IoT Events 감지기 모델 예제.

사용 사례

AWS IoT Events 에는 많은 용도가 있습니다. 다음은 몇 가지 사용 사례의 예입니다.

원격 디바이스 모니터링 및 유지 관리

원격으로 배포된 시스템 플릿을 모니터링하는 것은 어려울 수 있습니다. 특히 명확한 컨텍스트 없이 오작동이 발생하는 경우 더욱 그렇습니다. 한 머신이 작동을 멈추면 전체 처리 장치 또는 머신을 교체하

는 것일 수 있습니다. 하지만 이는 지속적이지 않습니다. 를 AWS IoT Events 사용하면 각 시스템의 여러 센서에서 메시지를 수신하여 시간이 지남에 따라 특정 문제를 진단하는 데 도움이 될 수 있습니다. 이제 전체 단위를 교체하는 대신 교체가 필요한 정확한 부분을 기술자에게 보내는 데 필요한 정보를 갖게 됩니다. 수백만 대의 기계를 사용할 경우 이를 통해 절감할 수 있는 비용이 최대 수백만 달러까지 늘어나 각 기계를 소유하거나 유지 관리하는 데 드는 총비용을 낮출 수 있습니다.

산업용 로봇 관리

시설에 로봇을 배포하여 패키지 이동을 자동화하면 효율성이 크게 향상될 수 있습니다. 비용을 최소화하기 위해 로봇에는 클라우드에 데이터를 보고하는 간단한 저비용 센서가 탑재될 수 있습니다. 그러나 수십 개의 센서와 수백 개의 작동 모드에서는 문제를 실시간으로 감지하는 것이 어려울 수 있습니다. 를 사용하면 클라우드에서이 센서 데이터를 처리하는 전문가 시스템을 구축하여 장애가 임박한 경우 기술 직원에게 자동으로 알리는 알림을 생성할 AWS IoT Events수 있습니다.

빌딩 자동화 시스템 추적

데이터 센터에서는 높은 온도와 낮은 습도를 모니터링하면 장비 장애를 방지하는 데 도움이 됩니다. 센서는 많은 제조업체에서 구매하는 경우가 많으며 각 유형에는 자체 관리 소프트웨어가 함께 제공됩니다. 그러나 다른 공급업체의 관리 소프트웨어가 호환되지 않아 문제를 감지하기 어려운 경우가 있습니다. AWS IoT Events를 사용하면 운영 분석가에게 히팅 및 쿨링 시스템의 문제를 장애 발생 전에 알리도록 알림을 설정할 수 있습니다. 이렇게 하면 수천 달러의 장비 교체 비용이 발생하고 잠재적인 수익 손실을 초래할 수 있는 예상치 못한 데이터 센터 가동 중단을 방지할 수 있습니다.

AWS IoT Events 지원 종료

신중한 고려 후 2026년 5월 20일부터 AWS IoT Events 서비스에 대한 지원을 종료하기로 결정했습니다. AWS IoT Events 는 2025년 5월 20일부터 더 이상 신규 고객을 받지 않습니다. 2025년 5월 20일 이전에 서비스에 가입한 계정이 있는 기존 고객은 AWS IoT Events 기능을 계속 사용할 수 있습니다. 2026년 5월 20일 이후에는 더 이상를 사용할 수 없습니다 AWS IoT Events.

이 페이지에서는 AWS IoT Events 고객이 비즈니스 요구 사항에 맞는 대체 솔루션으로 전환할 수 있는 지침과 고려 사항을 제공합니다.

> **Note**
>
> 이 가이드에 제시된 솔루션은 AWS IoT Events 기능에 대한 프로덕션 지원 대체가 아닌 예시로 사용하기 위한 것입니다. 비즈니스 요구 사항에 맞게 코드, 워크플로 및 관련 AWS 리소스를 사용자 지정합니다.

주제

에서 마이그레이션할 때 고려 사항 AWS IoT Events

- 각 구성 요소에 대해 최소 권한이 있는 IAM 역할 사용, 저장 및 전송 중 데이터 암호화를 비롯한 보안 모범 사례를 구현합니다. 자세한 내용은 IAM 사용 설명서의 IAM의 보안 모범 사례를 참조하세요.
- 데이터 수집 요구 사항에 따라 Kinesis 스트림의 샤드 수를 고려합니다. Kinesis 샤드에 대한 자세한 내용은 Amazon Kinesis Data Streams 개발자 안내서의 Amazon Kinesis Data Streams 용어 및 개념을 참조하세요 Amazon Kinesis.
- 지표 및 로그에 CloudWatch를 사용하여 포괄적인 모니터링 및 디버깅을 설정합니다. 자세한 내용은 Amazon CloudWatch 사용 설명서의 CloudWatch란 무엇입니까?를 참조하세요. Amazon CloudWatch
- 처리가 반복적으로 실패하는 메시지를 관리하는 방법, 재시도 정책 구현, 문제가 되는 메시지를 격리하고 분석하는 프로세스 설정 등 오류 처리의 구조를 고려합니다.
- AWS 요금 계산기를 사용하여 특정 사용 사례에 대한 비용을 추정합니다.

에서 감지기 모델의 마이그레이션 절차 AWS IoT Events

이 섹션에서는 마이그레이션할 때 유사한 감지기 모델 기능을 제공하는 대체 솔루션에 대해 설명합니다 AWS IoT Events.

AWS IoT Core 규칙을 통해 데이터 수집을 다른 AWS 서비스의 조합으로 마이그레이션할 수 있습니다. BatchPutMessage API를 통한 데이터 수집 대신 데이터를 AWS IoT Core MQTT 주제로 라우팅할 수 있습니다.

이 마이그레이션 접근 방식은 AWS IoT Core MQTT 주제를 IoT 데이터의 진입점으로 활용하여 직접 입력을 대체합니다 AWS IoT Events. MQTT 주제는 몇 가지 주요 이유로 선택됩니다. MQTT는 업계에서 널리 사용되므로 IoT 디바이스와의 호환성이 광범위합니다. 이러한 주제에서는 수많은 디바이스의 대량 메시지를 처리하여 확장성을 보장할 수 있습니다. 또한 콘텐츠 또는 디바이스 유형에 따라 메시지를 라우팅하고 필터링할 수 있는 유연성도 제공합니다. 또한 AWS IoT Core MQTT 주제는 다른 AWS 서비스와 원활하게 통합되어 마이그레이션 프로세스를 용이하게 합니다.

MQTT 주제에서 Amazon Kinesis Data Streams, 함수, Amazon DynamoDB 테이블 및 Amazon EventBridge 일정을 결합한 AWS Lambda 아키텍처로 데이터가 흐릅니다. 이러한 서비스 조합은 이전에에서 제공한 기능을 복제하고 개선 AWS IoT Events하여 IoT 데이터 처리 파이프라인에 대한 유연성과 제어를 강화합니다.

아키텍처 비교

현재 AWS IoT Events 아키텍처는 AWS IoT Core 규칙과 `BatchPutMessage` API를 통해 데이터를 수집합니다. 이 아키텍처는 데이터 수집 및 이벤트 게시 AWS IoT Core 에를 사용하며, 메시지는 AWS IoT Events 입력을 통해 상태 로직을 정의하는 감지기 모델로 라우팅됩니다. IAM 역할은 필요한 권한을 관리합니다.

새로운 솔루션은 데이터 수집을 AWS IoT Core 위해를 유지합니다(이제 전용 입력 및 출력 MQTT 주제 사용). 데이터 파티셔닝을 위한 Kinesis Data Streams와 상태 로직을 위한 평가자 Lambda 함수를 소개합니다. 이제 디바이스 상태가 DynamoDB 테이블에 저장되고 향상된 IAM 역할이 이러한 서비스 전반의 권한을 관리합니다.

용도	Solution	차이
데이터 수집 - IoT 디바이스에서 데이터를 수신합니다.	AWS IoT Core	이제 두 가지 MQTT 주제가 필요합니다. 하나는 디바이스 데이터를 수집하기 위한 주제이고 다른 하나는 출력 이벤트를 게시하기 위한 주제입니다.

용도	Solution	차이
메시지 방향 - 수신 메시지를 적절한 서비스로 라우팅합니다.	AWS IoT Core 메시지 라우팅 규칙	동일한 라우팅 기능을 유지하지만 이제 대신 Kinesis Data Streams로 메시지를 전달합니다. AWS IoT Events
데이터 처리 - 수신 데이터 스트림을 처리하고 구성합니다.	Kinesis Data Streams	AWS IoT Events 입력 기능을 대체하여 데이터 수집을 메시지 처리를 위한 디바이스 ID 파티셔닝으로 제공합니다.
로직 평가 - 상태 변경을 처리하고 작업을 트리거합니다.	평가자 Lambda	AWS IoT Events 감지기 모델을 대체하여 시각적 워크플로 대신 코드를 통해 사용자 지정 가능한 상태 로직 평가 제공
상태 관리 - 디바이스 상태를 유지합니다.	DynamoDB 테이블	디바이스 상태의 영구 스토리지를 제공하여 내부 AWS IoT Events 상태 관리를 대체하는 새로운 구성 요소
보안 - 서비스 권한을 관리합니다.	IAM 역할	업데이트된 권한에는 이제 기존 권한 외에도 Kinesis Data Streams, DynamoDB 및 EventBridge에 대한 액세스 AWS IoT Core 가 포함됩니다.

1단계: (선택 사항) AWS IoT Events 감지기 모델 구성 내보내기

새 리소스를 생성하기 전에 AWS IoT Events 감지기 모델 정의를 내보냅니다. 여기에는 이벤트 처리 로직이 포함되며 새 솔루션을 구현하기 위한 기록 참조 역할을 할 수 있습니다.

Console

를 사용하여 다음 단계를 AWS IoT Events AWS Management Console수행하여 감지기 모델 구성을 내보냅니다.

를 사용하여 감지기 모델을 내보내려면 AWS Management Console

1. AWS IoT Events 콘솔에 로그인합니다.
2. 왼쪽 탐색 창에서 [Detector models(감지기 모델)]를 선택합니다.
3. 내보낼 감지기 모델을 선택합니다.

4. 내보내기를 선택합니다. 출력과 관련된 정보 메시지를 읽은 다음 내보내기를 다시 선택합니다.
5. 내보내려는 각 감지기 모델에 대해 프로세스를 반복합니다.

감지기 모델의 JSON 출력이 포함된 파일이 브라우저의 다운로드 폴더에 추가됩니다. 필요에 따라 각 감지기 모델 구성을 저장하여 기록 데이터를 보존할 수 있습니다.

AWS CLI

를 사용하여 다음 명령을 AWS CLI실행하여 감지기 모델 구성을 내보냅니다.

를 사용하여 감지기 모델을 내보내려면 AWS CLI

1. 계정의 모든 감지기 모델을 나열합니다.

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

2. 각 감지기 모델에 대해 다음을 실행하여 구성을 내보냅니다.

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

3. 각 감지기 모델의 출력을 저장합니다.

2단계 - IAM 역할 생성

의 기능을 복제할 수 있는 권한을 제공하는 IAM 역할을 생성합니다 AWS IoT Events. 이 예제의 역할은 상태 관리를 위한 DynamoDB, 예약을 위한 EventBridge, 데이터 수집을 위한 Kinesis Data Streams, 메시지 게시를 AWS IoT Core 위한 및 로깅을 위한 CloudWatch에 대한 액세스 권한을 부여합니다. 이러한 서비스는 함께를 대체하는 역할을 합니다 AWS IoT Events.

1. 다음 권한을 가진 IAM 역할을 만듭니다. IAM 역할 생성에 대한 자세한 지침은 IAM 사용 설명서의 AWS 서비스에 권한을 위임할 역할 생성을 참조하세요.

   JSON

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

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

```
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogStream",
                "logs:PutLogEvents"
            ],
            "Resource": [
            "arn:aws:logs:us-east-1:123456789012:log-group:/aws/lambda/your-
lambda:*"
            ]
        }
    ]
}
```

2. 다음 IAM 역할 신뢰 정책을 추가합니다. 신뢰 정책은 지정된 AWS 서비스가 필요한 작업을 수행할 수 있도록 IAM 역할을 수임하도록 허용합니다. IAM 신뢰 정책 생성에 대한 자세한 지침은 IAM 사용 설명서의 사용자 지정 신뢰 정책을 사용하여 역할 생성을 참조하세요.

   JSON

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

3단계: Amazon Kinesis Data Streams 생성

AWS Management Console 또는를 사용하여 Amazon Kinesis Data Streams를 생성합니다 AWS CLI.

Console

를 사용하여 Kinesis 데이터 스트림을 생성하려면 Amazon Kinesis Data Streams 개발자 안내서의 데이터 스트림 생성 페이지에 있는 절차를 AWS Management Console따르세요.

디바이스 수와 메시지 페이로드 크기에 따라 샤드 수를 조정합니다.

AWS CLI

를 사용하여 Amazon Kinesis Data Streams를 AWS CLI생성하여 디바이스에서 데이터를 수집하고 분할합니다.

Kinesis Data Streams는이 마이그레이션에서의 데이터 수집 기능을 대체하는 데 사용됩니다 AWS IoT Events. IoT 디바이스에서 실시간 스트리밍 데이터를 수집, 처리 및 분석하는 확장 가능하고 효율적인 방법을 제공하는 동시에 유연한 데이터 처리 및 다른 AWS 서비스와의 통합을 제공합니다.

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

디바이스 수와 메시지 페이로드 크기에 따라 샤드 수를 조정합니다.

4단계: MQTT 메시지 라우팅 규칙 생성 또는 업데이트

새 MQTT 메시지 라우팅 규칙을 생성하거나 기존 규칙을 업데이트할 수 있습니다.

Console

1. 새 MQTT 메시지 라우팅 규칙이 필요한지 또는 기존 규칙을 업데이트할 수 있는지 확인합니다.
2. AWS IoT Core 콘솔을 엽니다.
3. 탐색 창에서 메시지 라우팅을 선택한 다음 규칙을 선택합니다.
4. 관리 섹션에서 메시지 라우팅을 선택한 다음 규칙을 선택합니다.
5. 규칙 생성을 선택합니다.
6. 규칙 속성 지정 페이지에서 AWS IoT Core 규칙 이름에 규칙 이름을 입력합니다. 규칙 설명 - 선택 사항에서 이벤트를 처리하고 있음을 식별하여 Kinesis Data Streams에 전달하는 설명을 입력합니다.
7. SQL 문 구성 페이지에서 SQL 문에 다음을 입력한 후 다음을 **`SELECT * FROM 'your-database'`**선택합니다.
8. 규칙 작업 연결 페이지의 규칙 작업에서 kinesis를 선택합니다.

9. 스트림에 대한 Kinesis 스트림을 선택합니다. 파티션 키에 **your-instance-id**을 입력합니다. IAM 역할에 적합한 역할을 선택한 다음 규칙 작업 추가를 선택합니다.

자세한 내용은 디바이스 데이터를 다른 서비스로 라우팅하는 AWS IoT 규칙 생성을 참조하세요.

AWS CLI

1. 다음 콘텐츠를 포함하는 JSON 파일을 생성합니다. 이 JSON 구성 파일은 인스턴스 ID를 파티션 키로 사용하여 주제에서 모든 메시지를 선택하고 지정된 Kinesis 스트림으로 전달하는 AWS IoT Core 규칙을 정의합니다.

```
{
    "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. 를 사용하여 MQTT 주제 규칙을 생성합니다 AWS CLI. 이 단계에서는 AWS CLI 를 사용하여 `events_rule.json` 파일에 정의된 구성을 사용하여 AWS IoT Core 주제 규칙을 생성합니다.

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

5단계: 대상 MQTT 주제의 엔드포인트 가져오기

대상 MQTT 주제를 사용하여 주제가 발신 메시지를 게시하는 위치를 구성하고 이전에 처리한 기능을 바꿉니다 AWS IoT Events. 엔드포인트는 AWS 계정 및 리전에 고유합니다.

Console

1. AWS IoT Core 콘솔을 엽니다.
2. 왼쪽 탐색 패널의 연결 섹션에서 도메인 구성을 선택합니다.
3. iot:Data-ATS 도메인 구성을 선택하여 구성의 세부 정보 페이지를 엽니다.
4. 도메인 이름 값을 복사합니다. 이 값은 엔드포인트입니다. 엔드포인트 값은 이후 단계에서 필요하므로 저장합니다.

AWS CLI

다음 명령을 실행하여 계정에 대한 발신 메시지를 게시하기 위한 AWS IoT Core 엔드포인트를 가져옵니다.

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

6단계: Amazon DynamoDB 테이블 생성

Amazon DynamoDB 테이블은의 상태 관리 기능을 대체하여 새로운 솔루션 아키텍처에서 디바이스의 상태와 감지기 모델 로직을 지속하고 관리할 수 있는 확장 가능하고 유연한 방법을 AWS IoT Events제공합니다.

Console

Amazon DynamoDB 테이블을 생성하여 감지기 모델의 상태를 유지합니다. 자세한 내용은 Amazon DynamoDB 개발자 안내서의 DynamoDB에서 테이블 생성을 참조하세요. DynamoDB

테이블 세부 정보는 다음을 사용합니다.

- 테이블 이름에 선택한 테이블 이름을 입력합니다.
- 파티션 키에 자체 인스턴스 ID를 입력합니다.
- 테이블 설정에 기본 설정을 사용할 수 있습니다.

AWS CLI

다음 명령을 실행하여 DynamoDB 테이블을 생성합니다.

```
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 \
```

7단계: AWS Lambda 함수 생성(콘솔)

Lambda 함수는의 감지기 모델 평가 로직을 대체하는 코어 처리 엔진 역할을 합니다 AWS IoT Events. 이 예제에서는 다른 AWS 서비스와 통합되어 수신 데이터를 처리하고, 상태를 관리하고, 정의된 규칙에 따라 작업을 트리거합니다.

NodeJS 런타임을 사용하여 Lambda 함수를 생성합니다. 하드 코딩된 상수를 대체하여 다음 코드 조각을 사용합니다.

1. AWS Lambda console을 엽니다.
2. 함수 생성을 선택합니다.
3. 함수 이름의 이름을 입력합니다.
4. NodeJS 22.x를 런타임으로 선택합니다.
5. 기본 실행 역할 변경 드롭다운에서 기존 역할 사용을 선택한 다음 이전 단계에서 생성한 IAM 역할을 선택합니다.
6. 함수 생성을 선택합니다.
7. 하드 코딩된 상수를 교체한 후 다음 코드 조각을 붙여 넣습니다.
8. 함수가 생성된 후 코드 탭에서 다음 코드 예제를 붙여넣고 **your-destination-endpoint** 엔드포인트를 사용자 고유의 것으로 바꿉니다.

```
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
        ]
    );
```

8단계: Amazon Kinesis Data Streams 트리거 추가

AWS Management Console 또는를 사용하여 Lambda 함수에 Kinesis Data Streams 트리거를 추가합니다 AWS CLI.

Lambda 함수에 Kinesis Data Streams 트리거를 추가하면 데이터 수집 파이프라인과 처리 로직 간의 연결이 설정되므로가 입력을 AWS IoT Events 처리하는 방식과 마찬가지로 들어오는 IoT 데이터 스트림을 자동으로 평가하고 이벤트에 실시간으로 대응할 수 있습니다.

Console

자세한 내용은 AWS Lambda 개발자 안내서의 이벤트 소스 매핑 생성을 참조하여 Lambda 함수를 호출하세요.

이벤트 소스 매핑 세부 정보에 다음을 사용합니다.

- 함수 이름에에 사용된 Lambda 이름을 입력합니다7단계: AWS Lambda 함수 생성(콘솔).
- 소비자 - 선택 사항에서 Kinesis 스트림의 ARN을 입력합니다.
- 배치 크기에 **10**을 입력합니다.

AWS CLI

다음 명령을 실행하여 Lambda 함수 트리거를 생성합니다.

```
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
```

9단계: 데이터 수집 및 출력 기능 테스트(AWS CLI)

감지기 모델에서 정의한 내용을 기반으로 MQTT 주제에 페이로드를 게시합니다. 다음은 구현을 테스트하기 위한 MQTT 주제의 `your-topic-name` 페이로드 예제입니다.

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

다음(또는 유사한) 콘텐츠가 포함된 주제에 게시된 MQTT 메시지가 표시되어야 합니다.

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

의 AWS IoT SiteWise 경보에 대한 마이그레이션 절차 AWS IoT Events

이 섹션에서는 마이그레이션할 때 유사한 경보 기능을 제공하는 대체 솔루션에 대해 설명합니다 AWS IoT Events.

AWS IoT Events 경보를 사용하는 AWS IoT SiteWise 속성의 경우 CloudWatch 경보를 사용하여 솔루션으로 마이그레이션할 수 있습니다. 이 접근 방식은 설정된 SLAs와 이상 탐지 및 그룹화된 경보와 같은 추가 기능을 통해 강력한 모니터링 기능을 제공합니다.

아키텍처 비교

AWS IoT SiteWise 속성에 대한 현재 AWS IoT Events 경보 구성을 사용하려면 AWS IoT SiteWise 사용 설명서의 `AssetModelCompositeModels`에서 외부 경보 정의에 AWS IoT SiteWise 설명된 대로 자산 모델에서를 생성해야 합니다. 새 솔루션의 수정은 일반적으로 콘솔을 AWS IoT Events 통해 관리됩니다.

새로운 솔루션은 CloudWatch 경보를 활용하여 경보 관리를 제공합니다. 이 접근 방식은 AWS IoT SiteWise 알림을 사용하여 속성 데이터 포인트를 AWS IoT Core MQTT 주제에 게시한 다음 Lambda

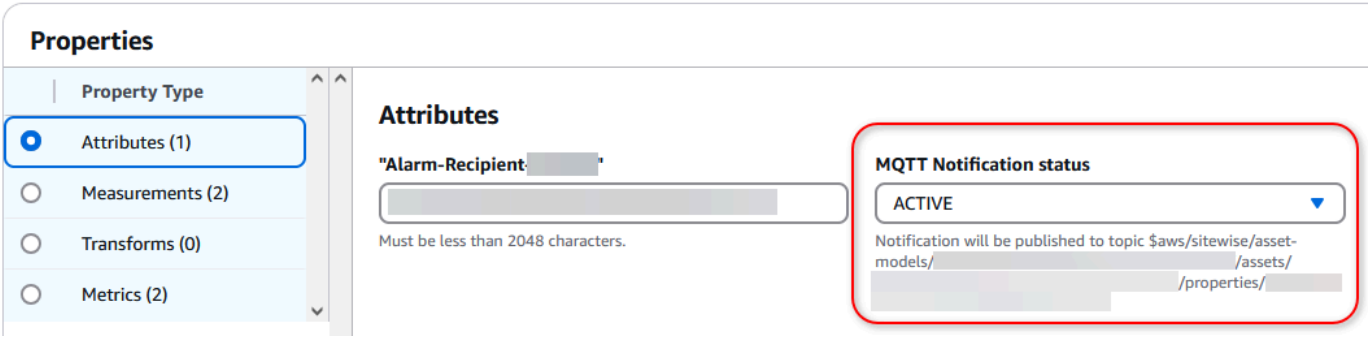
함수로 처리합니다. 함수는 이러한 알림을 CloudWatch 지표로 변환하여 CloudWatch의 경보 프레임워크를 통해 경보 모니터링을 활성화합니다.

용도	Solution	차이
데이터 소스 -의 속성 데이터 AWS IoT SiteWise	AWS IoT SiteWise MQTT 알림	AWS IoT SiteWise 속성의 MQTT 알림으로 직접 IoT Events 통합 대체
데이터 처리 - 속성 데이터를 변환합니다.	Lambda 함수	AWS IoT SiteWise 속성 알림을 처리하고 이를 CloudWatch 지표로 변환합니다.
경보 평가 - 지표를 모니터링하고 경보를 트리거합니다.	Amazon CloudWatch 경보	AWS IoT Events 경보를 CloudWatch 경보로 대체하여 이상 탐지와 같은 추가 기능을 제공합니다.
통합 -와 연결 AWS IoT SiteWise	AWS IoT SiteWise 외부 경보	CloudWatch 경보를 다시 외부 경보 AWS IoT SiteWise 로 가져오는 선택적 기능

1단계: 자산 속성에서 MQTT 알림 활성화

AWS IoT SiteWise 경보에 AWS IoT Events 통합을 사용하는 경우 모니터링할 각 속성에 대해 MQTT 알림을 켤 수 있습니다.

1. 자산 모델의 속성을 편집하는 각 단계까지 진행 중인 자산에 대한 경보 구성 AWS IoT SiteWise 절차를 따릅니다.
2. 마이그레이션할 각 속성에 대해 MQTT 알림 상태를 ACTIVE로 변경합니다.



3. 수정된 각 경보 속성에 대해 경보가 게시하는 주제 경로를 기록해 둡니다.

자세한 내용은 다음 설명서 리소스를 참조하세요.

- AWS IoT SiteWise 사용 설명서의 MQTT 주제에서 자산 속성을 이해합니다.
- AWS IoT 개발자 안내서의 MQTT 주제.

2단계: AWS Lambda 함수 생성

MQTT 주제에서 게시한 TQV 배열을 읽기 위한 Lambda 함수를 생성하고 개별 값을 CloudWatch에 게시합니다. 이 Lambda 함수를 AWS IoT Core 메시지 규칙에서 트리거할 대상 작업으로 사용합니다.

1. AWS Lambda console을 엽니다.
2. 함수 생성을 선택합니다.
3. 함수 이름의 이름을 입력합니다.
4. NodeJS 22.x를 런타임으로 선택합니다.
5. 기본 실행 역할 변경 드롭다운에서 기존 역할 사용을 선택한 다음 이전 단계에서 생성한 IAM 역할을 선택합니다.

> **Note**
>
> 이 절차에서는 감지기 모델을 이미 마이그레이션했다고 가정합니다. IAM 역할이 없는 경우 섹션을 참조하세요???.

6. 함수 생성을 선택합니다.
7. 하드 코딩된 상수를 교체한 후 다음 코드 조각을 붙여 넣습니다.

```
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)}')
        }
```

3단계: AWS IoT Core 메시지 라우팅 규칙 생성

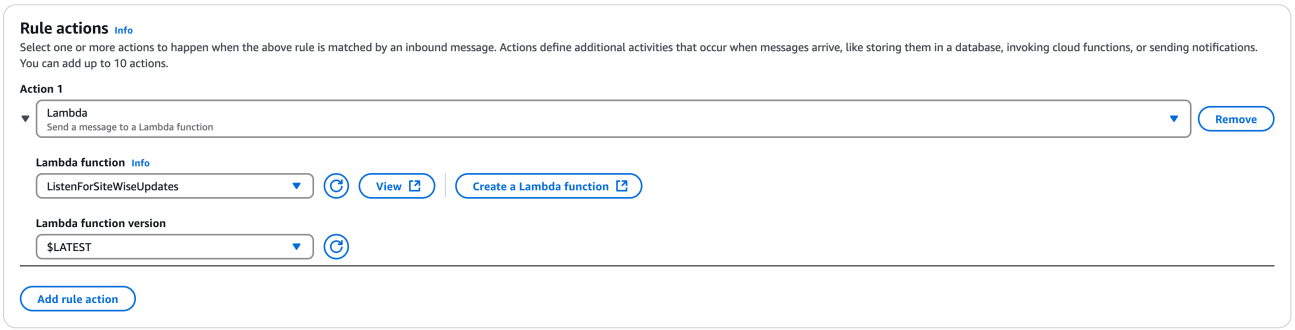
- 자습서: MQTT 메시지 재게시 절차에 따라 메시지가 표시되면 다음 정보를 입력합니다.

  a. 메시지 라우팅 규칙의 이름을 지정합니다`SiteWiseToCloudwatchAlarms`.

  b. 쿼리의 경우 다음을 사용할 수 있습니다.

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

  c. 규칙 작업에서 Lambda 작업을 선택하여에서 생성된 데이터를 CloudWatch AWS IoT SiteWise 로 전송합니다. 예제:



4단계: CloudWatch 지표 보기

에서 이전에 선택한 속성 AWS IoT SiteWise인에 데이터를 수집하면 1단계: 자산 속성에서 MQTT 알림 활성화는에서 생성한 Lambda 함수로 데이터를 라우팅합니다2단계: AWS Lambda 함수 생성. 이 단계에서는 Lambda가 지표를 CloudWatch로 전송하는지 확인할 수 있습니다.

1. CloudWatch AWS Management Console를 엽니다.
2. 왼쪽 탐색 창에서 지표를 선택한 다음 모든 지표를 선택합니다.
3. 경보의 URL을 선택하여 엽니다.
4. 소스 탭에서 CloudWatch 출력은이 예제와 비슷합니다. 이 소스 정보는 지표 데이터가 CloudWatch로 공급되고 있음을 확인합니다.

```
{
    "view": "timeSeries",
    "stacked": false,
    "metrics": [
```

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

5단계: CloudWatch 경보 생성

Amazon CloudWatch 사용 설명서의 정적 임계값을 기반으로 CloudWatch 경보 생성 절차에 따라 각 관련 지표에 대한 경보를 생성합니다. Amazon CloudWatch

> **Note**
>
> Amazon CloudWatch의 경보 구성에는 여러 옵션이 있습니다. CloudWatch 경보에 대한 자세한 내용은 Amazon CloudWatch 사용 설명서의 Amazon CloudWatch 경보 사용을 참조하세요. Amazon CloudWatch

6단계: (선택 사항) CloudWatch 경보를 로 가져오기 AWS IoT SiteWise

CloudWatch 경보 작업 및 Lambda를 AWS IoT SiteWise 사용하여 데이터를 로 다시 보내도록 CloudWatch 경보를 구성할 수 있습니다. 이 통합을 통해 SiteWise Monitor 포털에서 경보 상태 및 속성을 볼 수 있습니다.

1. 외부 경보를 자산 모델의 속성으로 구성합니다. 자세한 내용은 AWS IoT SiteWise 사용 설명서의 에서 외부 경보 정의를 AWS IoT SiteWise 참조하세요.
2. AWS IoT SiteWise 사용 설명서의 BatchPutAssetPropertyValue API를 사용하여 경보 데이터를 로 전송하는 Lambda 함수를 생성합니다 AWS IoT SiteWise.
3. 경보 상태가 변경될 때 Lambda 함수를 호출하도록 CloudWatch 경보 작업을 설정합니다. 자세한 내용은 Amazon CloudWatch 사용 설명서의 경보 작업 섹션을 참조하세요.

설 AWS IoT Events정

이 섹션에서는 AWS 계정 생성 AWS IoT Events, 필요한 권한 구성, 리소스에 대한 액세스 관리 역할 설정 등 설정에 대한 가이드를 제공합니다.

주제

설정 AWS 계정

에 가입 AWS 계정

이 없는 경우 다음 단계를 AWS 계정완료하여 생성합니다.

에 가입하려면 AWS 계정

1. https://portal.aws.amazon.com/billing/signup을 엽니다.
2. 온라인 지시 사항을 따르세요.

   등록 절차 중 전화 또는 텍스트 메시지를 받고 전화 키패드로 확인 코드를 입력하는 과정이 있습니다.

   에 가입하면 AWS 계정AWS 계정 루트 사용자이 생성됩니다. 루트 사용자에게는 계정의 모든 AWS 서비스 및 리소스에 액세스할 권한이 있습니다. 보안 모범 사례는 사용자에게 관리 액세스 권한을 할당하고, 루트 사용자만 사용하여 루트 사용자 액세스 권한이 필요한 작업을 수행하는 것입니다.

AWS 는 가입 프로세스가 완료된 후 확인 이메일을 보냅니다. 언제든지 https://aws.amazon.com/으로 이동하고 내 계정을 선택하여 현재 계정 활동을 확인하고 계정을 관리할 수 있습니다.

관리자 액세스 권한이 있는 사용자 생성

에 가입한 후 일상적인 작업에 루트 사용자를 사용하지 않도록 관리 사용자를 AWS 계정보호 AWS IAM Identity Center, AWS 계정 루트 사용자활성화 및 생성합니다.

보안 AWS 계정 루트 사용자

1. 루트 사용자를 선택하고 AWS 계정 이메일 주소를 입력하여 계정 소유자AWS Management Console로에 로그인합니다. 다음 페이지에서 비밀번호를 입력합니다.

   루트 사용자를 사용하여 로그인하는 데 도움이 필요하면 AWS Sign-In 사용 설명서의 루트 사용자로 로그인을 참조하세요.
2. 루트 사용자의 다중 인증(MFA)을 활성화합니다.

   지침은 IAM 사용 설명서의 AWS 계정 루트 사용자(콘솔)에 대한 가상 MFA 디바이스 활성화를 참조하세요.

관리자 액세스 권한이 있는 사용자 생성

1. IAM Identity Center를 활성화합니다.

   지침은 AWS IAM Identity Center 사용 설명서의 AWS IAM Identity Center설정을 참조하세요.
2. IAM Identity Center에서 사용자에게 관리 액세스 권한을 부여합니다.

   를 자격 증명 소스 IAM Identity Center 디렉터리 로 사용하는 방법에 대한 자습서는 사용 AWS IAM Identity Center 설명서의 기본값으로 사용자 액세스 구성을 IAM Identity Center 디렉터리 참조하세요.

관리 액세스 권한이 있는 사용자로 로그인

- IAM IDentity Center 사용자로 로그인하려면 IAM Identity Center 사용자를 생성할 때 이메일 주소로 전송된 로그인 URL을 사용합니다.

  IAM Identity Center 사용자를 사용하여 로그인하는 데 도움이 필요하면 사용 설명서의 AWS 액세스 포털에 로그인을 참조하세요. AWS Sign-In

추가 사용자에게 액세스 권한 할당

1. IAM Identity Center에서 최소 권한 적용 모범 사례를 따르는 권한 세트를 생성합니다.

   지침은 AWS IAM Identity Center 사용 설명서의 Create a permission set를 참조하세요.
2. 사용자를 그룹에 할당하고, 그룹에 Single Sign-On 액세스 권한을 할당합니다.

   지침은 AWS IAM Identity Center 사용 설명서의 그룹 추가를 참조하세요.

에 대한 권한 설정 AWS IoT Events

를 안전하고 효과적으로 사용하려면 적절한 권한을 구현하는 것이 중요합니다 AWS IoT Events. 이 섹션에서는의 일부 기능을 사용하는 데 필요한 권한을 설명합니다 AWS IoT Events. AWS CLI 명령 또는 AWS Identity and Access Management (IAM) 콘솔을 사용하여 리소스에 액세스하거나에서 특정 함수를 수행하기 위한 역할 및 관련 권한 정책을 생성할 수 있습니다 AWS IoT Events.

IAM 사용 설명서에는 AWS 리소스에 대한 액세스 권한을 안전하게 제어하는 방법에 대한 자세한 정보가 나와 있습니다. 에 대한 자세한 내용은 에 사용되는 작업, 리소스 및 조건 키를 AWS IoT Events AWS IoT Events참조하세요.

IAM 콘솔을 사용하여 역할 및 권한을 생성하고 관리하려면 IAM 자습서: IAM 역할을 사용하여 AWS 계정 간 액세스 권한 위임을 참조하세요.

> **Note**
>
> 키는 1~128자일 수 있으며 다음을 포함할 수 있습니다.
>
> - a-z 사이의 대문자 또는 소문자
> - 숫자(0-9)
> - 특수 문자 -, _, 또는 :.

에 대한 작업 권한 AWS IoT Events

AWS IoT Events 를 사용하면 다른 AWS 서비스를 사용하는 작업을 트리거할 수 있습니다. 이렇게 하려면 사용자를 대신하여 이러한 작업을 수행할 수 있는 AWS IoT Events 권한을 부여해야 합니다. 이 섹션에는 작업 목록과 리소스에서 이러한 모든 작업을 수행할 수 있는 권한을 부여하는 예시 정책이 포함되어 있습니다. 필요에 따라 *##* 및 *account-id* 참조를 변경하십시오. 가능하면 액세스할 수 있는 특정 리소스를 참조하도록 와일드카드(*) 도 변경해야 합니다. IAM 콘솔을 사용하여 정의한 Amazon SNS 알림을 전송할 수 있는 권한을 AWS IoT Events 에 부여할 수 있습니다.

AWS IoT Events 는 타이머를 사용하거나 변수를 설정할 수 있는 다음 작업을 지원합니다.

- `setTimer`로 타이머 생성.
- `resetTimer`로 타이머 재설정.
- `clearTimer`로 타이머 삭제.

- setVariable로 변수 생성.

AWS IoT Events 는 AWS 서비스를 사용할 수 있는 다음 작업을 지원합니다.

- iotTopicPublish로 MQTT 주제에 대한 메시지 게시.
- iotEvents로 AWS IoT Events 에 입력 값으로 데이터 전송.
- iotSiteWise로 AWS IoT SiteWise의 자산 속성에 데이터를 보냅니다.
- dynamoDB로 Amazon DynamoDB 테이블에 데이터 전송.
- dynamoDBv2로 Amazon DynamoDB 테이블에 데이터 전송.
- firehose Amazon Data Firehose 스트림으로 데이터를 전송합니다.
- lambda로 AWS Lambda 함수 호출.
- sns로 푸시 알림에 데이터 전송.
- sqs로 Amazon SQS 대기열에 데이터 전송.

Example정책

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iot:Publish",
            "Resource": "arn:aws:iot:us-east-1:123456789012:topic/*"
        },
        {
            "Effect": "Allow",
            "Action": "iotevents:BatchPutMessage",
            "Resource": "arn:aws:iotevents:us-east-1:123456789012:input/*"
        },
        {
            "Effect": "Allow",
            "Action": "iotsitewise:BatchPutAssetPropertyValue",
            "Resource": "*"
        },
        {
```

```
            "Effect": "Allow",
            "Action": "dynamodb:PutItem",
            "Resource": "arn:aws:dynamodb:us-east-1:123456789012:table/*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "firehose:PutRecord",
                "firehose:PutRecordBatch"
            ],
            "Resource": "arn:aws:firehose:us-east-1:123456789012:deliverystream/
*"
        },
        {
            "Effect": "Allow",
            "Action": "lambda:InvokeFunction",
            "Resource": "arn:aws:lambda:us-east-1:123456789012:function:*"
        },
        {
            "Effect": "Allow",
            "Action": "sns:Publish",
            "Resource": "arn:aws:sns:us-east-1:123456789012:*"
        },
        {
            "Effect": "Allow",
            "Action": "sqs:SendMessage",
            "Resource": "arn:aws:sqs:us-east-1:123456789012:*"
        }
    ]
}
```

에서 입력 데이터 보호 AWS IoT Events

감지기 모델에서 사용할 입력 데이터에 대한 액세스 권한을 부여할 수 있는 사람을 고려하는 것이 중요합니다. 전체 권한을 제한하고 싶지만 감지기 모델을 만들거나 업데이트할 수 있는 사용자 또는 엔티티가 있는 경우, 해당 사용자 또는 엔티티에 입력 라우팅을 업데이트할 수 있는 권한도 부여해야 합니다. 즉, `iotevents:UpdateDetectorModel` 및 `iotevents:CreateDetectorModel`에 대한 권한을 부여하는 것 외에 `iotevents:UpdateInputRouting` 권한도 부여해야 합니다.

Example

다음에서는 `iotevents:UpdateInputRouting`에 대한 권한 정책을 보여 줍니다.

JSON

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

와일드카드 “*”에 “Resource”를 입력하는 대신 Amazon 리소스 이름(ARN) 입력 목록을 지정하여 이 권한을 특정 입력으로 제한할 수 있습니다. 이를 통해 사용자 또는 개체가 생성하거나 업데이트한 감지기 모델에서 사용하는 입력 데이터에 대한 액세스를 제한할 수 있습니다.

에 대한 Amazon CloudWatch 로깅 역할 정책 AWS IoT Events

다음 정책 문서는 AWS IoT Events 가 사용자 대신 CloudWatch에 로그를 제출하도록 허용하는 역할 정책 및 신뢰 정책을 제공합니다.

역할 정책:

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogGroup",
                "logs:CreateLogStream",
                "logs:PutLogEvents",
                "logs:PutMetricFilter",
                "logs:PutRetentionPolicy",
```

```
                "logs:GetLogEvents",
                "logs:DeleteLogStream"
            ],
            "Resource": [
                "arn:aws:logs:*:*:*"
            ]
        }
    ]
}
```

신뢰 정책:

JSON

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

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

사용자가 다음과 같은 역할을 전달할 수 있도록 하는 IAM 권한 정책도 사용자에게 연결되어 있어야 합니다. 자세한 내용은 IAM 사용 설명서의 AWS 서비스에 역할을 전달할 수 있는 사용자 권한 부여를 참조하세요.

JSON

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

```
        {
            "Sid": "",
            "Effect": "Allow",
            "Action": [
                "iam:GetRole",
                "iam:PassRole"
            ],
            "Resource": "arn:aws:iam::123456789012:role/Role_To_Pass"
        }
    ]
}
```

다음 명령을 사용하여 CloudWatch 로그의 리소스 정책을 입력합니다. 이렇게 하면 AWS IoT Events 를 CloudWatch 스트림의 로그 이벤트로 넣을 수 있습니다.

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

다음 명령을 사용하여 로깅 옵션을 입력합니다. `roleArn`을 생성한 로그 역할로 바꿉니다.

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

에 대한 Amazon SNS 메시징 역할 정책 AWS IoT Events

Amazon SNS AWS IoT Events 와 통합하려면 안전하고 효율적인 알림 전송을 위해 신중한 권한 관리가 필요합니다. 이 가이드에서는가 Amazon SNS 주제에 메시지를 AWS IoT Events 게시할 수 있도록 IAM 역할 및 정책을 구성하는 프로세스를 안내합니다.

다음 정책 문서는 AWS IoT Events 가 사용자 대신 SNS 메시지를 제출하도록 허용하는 역할 정책 및 신뢰 정책을 제공합니다.

역할 정책:

JSON

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

신뢰 정책:

JSON

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

AWS IoT Events 콘솔 시작하기

이 섹션에서는 AWS IoT Events 콘솔을 사용하여 입력 및 감지기 모델을 생성하는 방법을 보여줍니다. 엔진의 두 가지 상태, 즉 정상 상태와 과압 상태를 모델링합니다. 엔진에서 측정된 압력이 특정 임계값을 초과하면 모델이 정상 상태에서 과압 상태로 전환됩니다. 그런 다음 Amazon SNS 메시지를 보내 기술자에게 상태를 알립니다. 압력 판독값이 3회 연속으로 다시 임계값 아래로 떨어지면 모델이 정상 상태로 돌아가고 또 다른 Amazon SNS 메시지로 확인 메시지를 보냅니다.

비선형 회복 단계 또는 비정상적인 압력 판독값의 경우, 압력 임계값 미만으로 3회 연속 측정값을 확인하여 과압 또는 정상 메시지의 끊김 현상을 방지합니다.

콘솔에서는 사용자가 지정할 수 있는 미리 만들어진 여러 개의 감지기 모델 템플릿도 찾을 수 있습니다. 콘솔을 사용하여 다른 사용자가 작성한 감지기 모델을 가져오거나 감지기 모델을 내보내고 다른 AWS 리전에서 사용할 수도 있습니다. 감지기 모델을 가져오는 경우 새 리전에 필요한 입력을 생성하거나 다시 생성하고 사용된 역할 ARN을 업데이트해야 합니다.

AWS IoT Events 콘솔을 사용하여 다음에 대해 알아봅니다.

입력 정의

디바이스와 프로세스를 모니터링하려면 원격 측정 데이터를 AWS IoT Events로 보낼 방법이 있어야 합니다. 이는 메시지를 입력으로 전송하여 수행됩니다 AWS IoT Events. 여러 가지 방법으로 이 작업을 수행할 수 있습니다.

- BatchPutMessage 작업을 사용합니다.
- 에서 메시지 데이터를 전달하는 AWS IoT 규칙 엔진에 대한 AWS IoT Events 작업 규칙을 AWS IoT Core작성합니다 AWS IoT Events. 입력 내용을 이름으로 식별해야 합니다.
- 에서 CreateDataset 작업을 AWS IoT 분석사용하여 로 데이터 세트를 생성합니다`contentDeliveryRules`. 이러한 규칙은 데이터 세트 콘텐츠가 자동으로 전송되는 AWS IoT Events 입력을 지정합니다.

디바이스에서 이러한 방식으로 데이터를 전송하려면 먼저 하나 이상의 입력을 정의해야 합니다. 이렇게 하려면 각 입력에 이름을 지정하고 입력이 모니터링하는 수신 메시지 데이터의 필드를 지정하십시오.

감지기 모델 생성

상태를 사용하여 감지기 모델(디바이스 또는 프로세스의 모델)을 정의합니다. 각 상태에 대해 수신되는 입력을 평가하여 중요 이벤트를 탐지하는 조건부(부울) 논리를 정의합니다. 감지기 모델이 이

벤트를 감지하면 상태를 변경하거나 다른 AWS 서비스를 사용하여 사용자 지정 빌드 또는 사전 정의된 작업을 시작할 수 있습니다. 특정 상태가 시작 또는 종료할 때 또한 선택적으로 특정 조건이 충족될 때 작업을 시작하는 추가 이벤트를 정의할 수 있습니다.

이 자습서에서는 모델이 특정 상태에 들어가거나 종료될 때 Amazon SNS 메시지를 작업으로 전송합니다.

디바이스 또는 프로세스 모니터링

여러 장치 또는 프로세스를 모니터링하는 경우 각 입력에 입력이 들어오는 특정 장치 또는 프로세스를 식별하는 필드를 지정하십시오. `CreateDetectorModel`의 `key` 필드를 참조하십시오. `key`로 식별된 입력 필드가 새 값을 인식하면 새 디바이스가 식별되고 감지기가 생성됩니다. 각 감지기는 감지기 모델의 인스턴스입니다. 새 감지기는 해당 감지기 모델이 업데이트되거나 삭제될 때까지 해당 장치에서 오는 입력에 계속 응답합니다.

단일 프로세스를 모니터링하는 경우(여러 장치 또는 하위 프로세스가 입력을 보내는 경우에도) 고유한 식별 `key` 필드를 지정하지 않습니다. 이 경우 모델은 첫 번째 입력이 도착하면 단일 감지기(인스턴스)를 생성합니다.

메시지를 감지기 모델에 입력으로 전송

디바이스에서 메시지를 보내거나 AWS IoT Events 감지기에 입력으로 처리하는 방법에는 여러 가지가 있으며, 이 경우 메시지에 추가 형식을 지정할 필요가 없습니다. 이 자습서에서는 AWS IoT 콘솔을 사용하여 메시지 데이터를 전달하는 AWS IoT 규칙 엔진에 대한 AWS IoT Events 작업 규칙을 작성합니다 AWS IoT Events.

이렇게 하려면 이름으로 입력을 식별하고 AWS IoT 콘솔을 계속 사용하여 입력으로 전달되는 메시지를 생성합니다 AWS IoT Events.

> **Note**
>
> 이 자습서에서는 AWS IoT Events 사용 사례에 대한 자습서의 예제에서 나타나는 콘솔을 사용하여 동일한 `input` 및 `detector model`을 생성합니다. 이 JSON 예제를 사용하면 자습서를 쉽게 이해할 수 있습니다.

주제

- 시작하기 위한 사전 조건 AWS IoT Events
- 에서 모델에 대한 입력 생성 AWS IoT Events

- 에서 감지기 모델 생성 AWS IoT Events
- 에서 감지기 모델을 테스트하기 위한 입력 전송 AWS IoT Events

시작하기 위한 사전 조건 AWS IoT Events

AWS 계정이 없는 경우 계정을 생성합니다.

1. 의 단계에 따라 계정 설정 및 권한이 적절한설 AWS IoT Events정지 확인합니다.
2. 두 개의 Amazon Simple Notification Service(SNS) 주제를 생성하십시오.

   이 자습서(및 해당 예제) 에서는 Amazon SNS 주제를 두 개 생성했다고 가정합니다. 이러한 주제의 ARN은 `arn:aws:sns:us-east-1:123456789012:underPressureAction` 또는 `arn:aws:sns:us-east-1:123456789012:pressureClearedAction`과 같이 표시됩니다. 이러한 값을 사용자가 생성한 Amazon SNS 주제의 ARN으로 대체하십시오. 자세한 내용은 Amazon Simple Notification Service Developer Guide를 참조하십시오.

   Amazon SNS 주제에 알림을 게시하는 대신, 감지기가 지정한 주제와 함께 MQTT 메시지를 보내도록 할 수 있습니다. 이 옵션을 사용하면 AWS IoT 코어 콘솔을 사용하여 해당 MQTT 주제로 전송된 메시지를 구독하고 모니터링하여 감지기 모델이 인스턴스를 생성하고 해당 인스턴스가 알림을 전송하고 있는지 확인할 수 있습니다. 감지기 모델에서 생성한 입력 또는 변수를 사용하여 런타임에 MQTT 주제 이름을 동적으로 정의할 수도 있습니다.
3. 에서 AWS 리전 지원하는를 선택합니다 AWS IoT Events. 자세한 내용은 AWS 일반 참조의 AWS IoT Events 섹션을 참조하세요. 도움이 필요하면 시작하기의에서 서비스 AWS Management Console 시작하기를 참조하세요 AWS Management Console.

에서 모델에 대한 입력 생성 AWS IoT Events

모델의 입력을 구성할 때는 디바이스 또는 프로세스가 상태를 보고하기 위해 보내는 샘플 메시지 페이로드가 포함된 파일을 수집하는 것이 좋습니다. 이러한 파일은 필요한 입력을 정의하는 데 도움이 됩니다.

이 섹션에 설명된 여러 방법을 통해 입력을 생성할 수 있습니다.

JSON 입력 파일 생성

1. 시작하려면 로컬 파일 시스템에 다음과 같은 내용으로 `input.json`(이)라는 이름을 지정한 파일을 만드십시오.

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

2. 이제 시작 `input.json` 파일이 생겼으므로 입력을 만들 수 있습니다. 입력을 생성하는 방법에는 두 가지가 있습니다. AWS IoT Events 콘솔의 탐색 창을 사용하여 입력을 생성할 수 있습니다. 또는 감지기 모델이 생성된 후 감지기 모델 내에서 입력을 생성할 수 있습니다.

입력 생성 및 구성

경보 모델 또는 감지기 모델에 대한 입력을 생성하는 방법을 알아봅니다.

1. AWS IoT Events 콘솔에 로그인하거나 새 AWS IoT Events 계정 생성 옵션을 선택합니다.
2. AWS IoT Events 콘솔의 왼쪽 상단 모서리에서 탐색 창을 선택하고 확장합니다.
3. 왼쪽 탐색 창에서 입력을 선택합니다.
4. 콘솔의 오른쪽 모서리에서 입력 생성을 선택합니다.
5. uniqueInputName을 제공합니다.
6. 선택 사항 - 입력에 대한 설명을 입력합니다.
7. JSON 파일을 업로드하려면의 개요에서 생성한 `input.json` 파일을 선택합니다JSON 입력 파일 생성. 입력 속성 선택은 입력한 속성 목록과 함께 나타납니다.
8. 입력 속성 선택에서 사용할 속성을 선택하고 생성을 선택합니다. 이 예제에서는 motorid 및 SensorData.pressure를 선택합니다.
9. 선택 사항 - 입력에 관련 태그를 추가합니다.

> **Note**
>
> AWS IoT Events 콘솔 에서 감지기 모델 내에 추가 입력을 생성할 수도 있습니다. 자세한 내용은 에서 감지기 모델 내에 입력 생성 AWS IoT Events 단원을 참조하십시오.

에서 감지기 모델 내에 입력 생성 AWS IoT Events

의 감지기 입력 AWS IoT Events 은 데이터 소스와 감지기 모델 간의 브리지 역할을 합니다. 감지기 입력은 이벤트 감지 및 자동화 기능을 지원하는 원시 데이터를 제공합니다 AWS IoT Events. 모델이 IoT 에코시스템의 실제 이벤트 및 조건에 정확하게 대응할 수 있도록 감지기 입력을 구성하는 방법을 알아봅니다.

이 섹션에서는 원격 측정 데이터 또는 메시지를 수신하기 위해 감지기 모델의 입력을 정의하는 방법을 보여줍니다.

감지기 모델의 입력을 정의하려면

1. AWS IoT Events 콘솔을 엽니다.
2. AWS IoT Events 콘솔에서 감지기 모델 생성을 선택합니다.
3. 새로 생성을 선택합니다.
4. 입력 생성을 선택합니다.
5. 입력에 InputName, 선택 사항인 설명을 입력하고 파일 업로드를 선택합니다. 표시되는 대화 상자에서의 개요에서 생성한 `input.json` 파일을 선택합니다JSON 입력 파일 생성.
6. 입력 속성 선택에서 사용할 속성을 선택하고 생성을 선택합니다. 이 예제에서는 motorId 및 sensorData.pressure를 선택합니다.

에서 감지기 모델 생성 AWS IoT Events

이 주제에서는 상태를 사용하여 감지기 모델(장비 또는 프로세스의 모델)을 정의합니다.

각 상태에 대해 수신되는 입력을 평가하여 중요 이벤트를 탐지하는 조건부(부울) 논리를 정의합니다. 이벤트가 감지되면 상태가 변경되고 추가 작업을 시작할 수 있습니다. 이러한 이벤트를 전환 이벤트라고 합니다.

또한 상태에서는 감지기가 해당 상태에 들어가거나 종료할 때마다, 혹은 입력을 받을 때마다 작업을 실행할 수 있는 이벤트(이를 `OnEnter`, `OnExit` 및 `OnInput` 이벤트라고 함)를 정의합니다. 작업은 이벤트의 조건부 로직이 `true`로 평가되는 경우에만 실행됩니다.

감지기 모델 생성

1. 첫 번째 감지기 상태가 생성되었습니다. 수정하려면 기본 편집 스페이스에서 State_1(이)라는 레이블이 있는 원을 선택하십시오.

2. 상태 창에서 상태 이름 및 OnEnter을 입력하고 이벤트 추가를 선택하십시오.
3. OnEnter 이벤트 추가 페이지에서 이벤트 이름과 이벤트 조건을 입력합니다. 이 예제에서는 `true`를 입력해 상태를 입력할 때 이벤트가 항상 시작됨을 나타냅니다.
4. 이벤트 작업에서 작업 추가를 선택합니다.
5. 이벤트 작업에서 다음을 수행합니다.

   a. 변수 설정을 선택합니다.

   b. 변수 작업의 경우 값 할당을 선택합니다.

   c. 변수 이름에는 설정할 변수의 이름을 입력합니다.

   d. 변수 값에 값 **`0`**(제로)을 입력합니다.
6. 저장을 선택합니다.

   정의한 것과 같은 변수는 감지기 모델의 모든 이벤트에서 설정(지정 값)될 수 있습니다. 감지기가 상태에 도달하고 해당 상태가 정의되거나 설정된 작업을 실행한 후에만 변수 값을 참조할 수 있습니다(예: 이벤트의 조건부 로직).
7. 상태 창에서 상태 옆의 X를 선택해 감지기 모델 팔레트로 돌아갑니다.
8. 두 번째 감지기 상태를 만들려면 감지기 모델 팔레트에서 상태를 선택하고 기본 편집 스페이스로 드래그하십시오. 그러면 `untitled_state_1` 제목이 붙은 상태가 만들어집니다.
9. 첫 번째 상태(보통)에서 일시 중지합니다. 상태 둘레에 화살표가 나타납니다.
10. 화살표를 클릭하여 첫 번째 상태에서 두 번째 상태로 드래그합니다. 첫 번째 상태에서 두 번째 상태로 이어지는 지시선(제목 없음)이 나타납니다.
11. 제목 없음 라인을 선택합니다. 전환 이벤트 창에서 이벤트 이름과 이벤트 트리거 로직을 입력합니다.
12. 전환 이벤트 창에서 작업 추가를 선택합니다.
13. 전환 이벤트 작업 추가 창에서 작업 추가를 선택합니다.
14. 작업 선택에서 변수 설정을 선택합니다.

   a. 변수 작업의 경우 값 할당을 선택합니다.

   b. 변수 이름에 변수의 이름을 입력합니다.

   c. 값 할당에 `$variable.pressureThresholdBreached + 3`과 같은 값을 입력합니다

   d. 저장을 선택합니다.
15. 두 번째 상태 untitled_state_1를 선택합니다.
16. 상태 창에서 주 이름을 입력하고 On Enter에 대해 이벤트 추가를 선택합니다.

17. OnEnter 이벤트 추가 페이지에서 이벤트 이름 및 이벤트 조건을 입력합니다. 작업 추가를 선택합니다.
18. 작업 선택에서 SNS 메시지 보내기를 선택합니다.
    a. SNS 주제에 사용할 Amazon SNS 주제의 대상 ARN을 입력합니다.
    b. 저장을 선택합니다.
19. 예제에 이벤트를 계속 추가하십시오.
    a. OnInput의 경우, 이벤트 추가를 선택하고 다음 이벤트 정보를 입력하고 저장합니다.

```
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. OnInput의 경우, 이벤트 추가를 선택하고 다음 이벤트 정보를 입력하고 저장합니다.

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

    c. OnExit의 경우, 이벤트 추가를 선택하고 생성한 Amazon SNS 주제의 ARN을 사용하여 다음 이벤트 정보를 입력하고 저장합니다.

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

20. 두 번째 상태(위험) 에서 일시 중지합니다. 상태 둘레에 화살표가 나타납니다.
21. 화살표를 클릭하여 두 번째 상태에서 첫 번째 상태로 드래그합니다. 제목 없음 레이블이 있는 지시선이 나타납니다.

22. 제목 없음 라인을 선택하고 전환 이벤트 창에서 다음 정보를 사용하여 이벤트 이름과 이벤트 트리거 로직을 입력합니다.

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

    트리거 로직에서 `$input` 값과 `$variable` 값을 테스트하는 이유에 대한 자세한 내용은 AWS IoT Events 감지기 모델 제한 사항 및 제한 사항의 변수 값 가용성 항목을 참조하십시오.

23. 시작 상태를 선택합니다. 기본적으로 이 상태는 감지기 모델을 생성할 때 생성되었습니다. 시작 창에서 대상 상태(예: 보통)를 선택합니다.
24. 그런 다음, 입력을 수신하도록 감지기 모델을 구성합니다. 오른쪽 상단 모서리에서 게시를 선택합니다.
25. 감지기 모델 게시 창에서 다음을 수행합니다.
    a. 감지기 모델 이름, 설명, 역할 이름을 입력합니다. 이 역할은 사용자를 위해 생성됩니다.
    b. 각 고유 키 값에 대한 감지기 만들기를 선택합니다. 고유한 역할을 생성하여 사용하려면 에 대한 권한 설정 AWS IoT Events에 나와 있는 단계를 따르고 여기에 역할로 입력합니다.
26. 감지기 생성 키의 경우, 이전에 정의한 입력 속성 중 하나의 이름을 선택합니다. 감지기 생성 키로 선택하는 속성은 각 메시지 입력에 있어야 하며 메시지를 보내는 각 디바이스마다 고유해야 합니다. 이 예제에서는 motorid 속성을 사용합니다.
27. [Save and publish(저장 및 게시)]를 선택합니다.

> **Note**
>
> 지정된 감지기 모델에 대해 생성되는 고유 감지기의 수는 전송된 입력 메시지를 기반으로 합니다. 감지기 모델이 생성되면 입력 속성에서 키가 선택됩니다. 이 키는 사용할 감지기 인스턴스를 결정합니다. (이 감지기 모델에서) 이전에 본 적이 없는 키일 경우, 새 감지기 인스턴스가 생성됩니다. 이전에 본 적이 있는 키일 경우, 이 키 값에 해당하는 기존 감지기 인스턴스를 사용합니다.

감지기 모델 정의의 백업 사본(JSON 내)을 만들어 감지기 모델을 재생성 또는 업데이트하거나 템플릿으로 사용하여 다른 감지기 모델을 생성할 수 있습니다.

콘솔 또는 다음 CLI 명령을 사용하여 수행할 수 있습니다. 필요한 경우 이전 단계에서 게시할 때 사용한 것과 일치하도록 감지기 모델 이름을 변경하십시오.

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

그러면 다음과 비슷한 내용이 포함된 파일(motorDetectorModel.json)이 생성됩니다.

```
{
    "detectorModel": {
        "detectorModelConfiguration": {
            "status": "ACTIVE",
            "lastUpdateTime": 1552072424.212,
            "roleArn": "arn:aws:iam::123456789012:role/IoTEventsRole",
            "creationTime": 1552072424.212,
            "detectorModelArn": "arn:aws:iotevents:us-
west-2:123456789012:detectorModel/motorDetectorModel",
            "key": "motorid",
            "detectorModelName": "motorDetectorModel",
            "detectorModelVersion": "1"
        },
        "detectorModelDefinition": {
            "states": [
                {
                    "onInput": {
                        "transitionEvents": [
                            {
                                "eventName": "Overpressurized",
                                "actions": [
                                    {
                                        "setVariable": {
                                            "variableName":
 "pressureThresholdBreached",
                                            "value":
 "$variable.pressureThresholdBreached + 3"
                                        }
                                    }
                                ],
                                "condition": "$input.PressureInput.sensorData.pressure
 > 70",
                                "nextState": "Dangerous"
                            }
                        ],
```

```
                        "events": []
                    },
                    "stateName": "Normal",
                    "onEnter": {
                        "events": [
                            {
                                "eventName": "init",
                                "actions": [
                                    {
                                        "setVariable": {
                                            "variableName":
"pressureThresholdBreached",
                                            "value": "0"
                                        }
                                    }
                                ],
                                "condition": "true"
                            }
                        ]
                    },
                    "onExit": {
                        "events": []
                    }
                },
                {
                    "onInput": {
                        "transitionEvents": [
                            {
                                "eventName": "Back to Normal",
                                "actions": [],
                                "condition": "$variable.pressureThresholdBreached <= 1
&& $input.PressureInput.sensorData.pressure <= 70",
                                "nextState": "Normal"
                            }
                        ],
                        "events": [
                            {
                                "eventName": "Overpressurized",
                                "actions": [
                                    {
                                        "setVariable": {
                                            "variableName":
"pressureThresholdBreached",
                                            "value": "3"
```

```
                                        }
                                    }
                                ],
                                "condition": "$input.PressureInput.sensorData.pressure
 > 70"
                            },
                            {
                                "eventName": "Pressure Okay",
                                "actions": [
                                    {
                                        "setVariable": {
                                            "variableName":
 "pressureThresholdBreached",
                                            "value":
 "$variable.pressureThresholdBreached - 1"
                                        }
                                    }
                                ],
                                "condition": "$input.PressureInput.sensorData.pressure
 <= 70"
                            }
                        ]
                    },
                    "stateName": "Dangerous",
                    "onEnter": {
                        "events": [
                            {
                                "eventName": "Pressure Threshold Breached",
                                "actions": [
                                    {
                                        "sns": {
                                            "targetArn": "arn:aws:sns:us-
west-2:123456789012:MyIoTButtonSNSTopic"
                                        }
                                    }
                                ],
                                "condition": "$variable.pressureThresholdBreached > 1"
                            }
                        ]
                    },
                    "onExit": {
                        "events": [
                            {
                                "eventName": "Normal Pressure Restored",
```

```
                                "actions": [
                                    {
                                        "sns": {
                                            "targetArn": "arn:aws:sns:us-
west-2:123456789012:IoTVirtualButtonTopic"
                                        }
                                    }
                                ],
                                "condition": "true"
                            }
                        ]
                    }
                }
            ],
            "initialStateName": "Normal"
        }
    }
}
```

에서 감지기 모델을 테스트하기 위한 입력 전송 AWS IoT Events

에서 원격 측정 데이터를 수신하는 방법에는 여러 가지가 있습니다 AWS IoT Events (참조에서 데이터를 수신하고 작업을 트리거하는 데 지원되는 작업 AWS IoT Events). 이 주제에서는 AWS IoT 콘솔에서 메시지를 AWS IoT Events 감지기에 입력으로 전달하는 AWS IoT 규칙을 생성하는 방법을 보여줍니다. AWS IoT 콘솔의 MQTT 클라이언트를 사용하여 테스트 메시지를 보낼 수 있습니다. 이 방법을 사용하면 디바이스가 AWS IoT 메시지 브로커를 사용하여 MQTT 메시지를 전송할 수 있을 AWS IoT Events 때에 원격 측정 데이터를 가져올 수 있습니다.

감지기 모델을 테스트하기 위한 입력을 보내려면

1. AWS IoT Core 콘솔을 엽니다. 왼쪽 탐색 창의 관리에서 메시지 라우팅을 선택한 다음 규칙을 선택합니다.
2. 오른쪽 상단의 규칙 만들기를 선택합니다.
3. 규칙 생성 페이지에서 다음 단계를 완료합니다.

   1. 1단계. 규칙 속성을 지정합니다. 다음 작업을 완료합니다.

      - 규칙 이름. `MyIoTEventsRule`과 같은 포털의 이름을 입력합니다.

> **Note**
>
> 공백은 사용할 수 없습니다.

- 규칙 설명. 이는 선택 사항입니다.
- 다음을 선택합니다.

2. 2단계. SQL 명령문 구성. 다음 작업을 완료합니다.
   - SQL 버전. 목록에서 적절한 옵션을 선택합니다.
   - SQL 명령문. **`SELECT *, topic(2) as motorid FROM 'motors/+/status'`**을 입력합니다.

   다음을 선택합니다.
3. 3단계. 규칙 작업을 첨부합니다. 규칙 작업 섹션에서 다음을 완료합니다.
   - 작업 1. IoT 이벤트를 선택합니다. 다음 필드가 표시됩니다.
     a. 입력 이름. 목록에서 적절한 옵션을 선택합니다. 암호가 표시되지 않으면 새로 고침을 선택합니다.

     새 입력을 생성하려면 IoT Events 입력 생성을 선택합니다. 다음 작업을 완료합니다.
     - 입력 이름. `PressureInput`을 입력합니다.
     - 설명. 이는 선택 사항입니다.
     - JSON 파일을 업로드하십시오. JSON 파일의 사본을 업로드하십시오. 파일이 없는 경우 이 화면에 샘플 파일로 연결되는 링크가 있습니다. 코드에는 다음이 포함됩니다.

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

     - 입력 속성을 선택합니다. 적절한 옵션을 선택합니다.
     - Tags)]를 선택합니다. 이는 선택 사항입니다.

     생성(Create)을 선택합니다.

규칙 생성 화면으로 돌아가 입력 이름 필드를 새로 고칩니다. 방금 생성한 입력을 선택합니다.

b. 배치 모드. 이는 선택 사항입니다. 페이로드가 메시지 배열인 경우 이 옵션을 선택하십시오.

c. 메시지 ID. 이는 선택 사항이며, 권장 사항은 아닙니다.

d. IAM 역할. 목록에서 적절한 역할을 선택합니다. 역할이 목록에 없는 경우 새 역할 생성을 선택합니다.

역할 이름을 입력하고 생성을 선택합니다.

다른 규칙을 추가하려면 규칙 추가 작업을 선택합니다.

- 오류 작업. 이 섹션은 선택 사항입니다. 작업을 추가하려면 오류 작업 추가를 선택하고 목록에서 적절한 작업을 선택합니다.

나타나는 필드를 모두 입력합니다.

- 다음을 선택합니다.

4. 4단계. 검토 및 생성. 화면에서 정보를 검토한 후 생성을 선택합니다.

4. 왼쪽 탐색 창의 테스트에서 MQTT 테스트 클라이언트를 선택합니다.

5. Publish to a topic(주제에 게시)을 선택합니다. 다음 작업을 완료합니다.

- 주제 이름. `motors/Fulton-A32/status`와 같이 메시지를 식별할 이름을 입력합니다.
- 메시지 페이로드. 다음을 입력합니다.

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

> **Note**
>
> 새 메시지를 게시할 때마다 `messageId`를 변경하십시오.

6. 게시의 경우, 주제를 동일하게 유지하되 페이로드의 `"pressure"` 값을 감지기 모델에서 지정한 임계값(예: **85**)보다 큰 값으로 변경하십시오.

7. 게시를 선택합니다.

생성된 감지기 인스턴스는 Amazon SNS 메시지를 생성하여 전송합니다. 압력 측정값이 압력 임계값(이 예제의 경우 70)보다 높거나 낮은 압력 측정값을 포함하는 메시지를 계속 보내면 감지기가 작동 중인 것을 확인할 수 있습니다.

이 예시에서는 압력 측정값이 임계값 미만인 메시지 3개를 전송하여 보통 상태로 다시 전환하고 과압 상태가 해결되었음을 나타내는 Amazon SNS 메시지를 수신해야 합니다. 보통 상태로 돌아오면 압력이 한도를 초과하는 메시지 하나가 감지기를 위험 상태로 전환하고 해당 상태를 나타내는 Amazon SNS 메시지를 전송합니다.

이제 간단한 입력 및 감지기 모델을 만들었으니 다음을 시도해 보십시오.

- 콘솔에서 더 많은 감지기 모델 예제(템플릿)를 참조하십시오.
- 의 단계에 따라 CLI를 사용하여 두 상태에 대한 AWS IoT Events 감지기 생성를 사용하여 입력 및 감지기 모델을 생성합니다. AWS CLI
- 이벤트에서 사용되는 이벤트 데이터를 필터링, 변환 및 처리하는 표현식에 대해 자세히 알아보십시오.
- 에서 데이터를 수신하고 작업을 트리거하는 데 지원되는 작업 AWS IoT Events에 대해 알아봅니다.
- 문제가 있으면 문제 해결 AWS IoT Events을 참조하십시오.

의 모범 사례 AWS IoT Events

다음 모범 사례를 따르면 AWS IoT Events의 이점을 최대한 활용할 수 있습니다.

주제

- AWS IoT Events 감지기 모델 개발 시 Amazon CloudWatch 로깅 활성화
- AWS IoT Events 콘솔에서 작업할 때 정기적으로 게시하여 감지기 모델을 저장합니다.

AWS IoT Events 감지기 모델 개발 시 Amazon CloudWatch 로깅 활성화

Amazon CloudWatch는 AWS 리소스와 AWS 실행 중인 애플리케이션을 실시간으로 모니터링합니다. CloudWatch를 사용하면 시스템 전체의 리소스 사용률, 애플리케이션 성능, 운영 상태를 파악할 수 있습니다. AWS IoT Events 감지기 모델을 개발하거나 디버깅할 때 CloudWatch는 어떤 작업이 AWS IoT Events 수행되고 있는지와 발생하는 오류를 파악할 수 있도록 도와줍니다.

CloudWatch 활성화

1. 아직 생성하지 않은 경우의 단계에 따라 CloudWatch 로그에 대한 권한 설정 AWS IoT Events를 생성하고 관리할 수 있는 권한을 부여하는 연결된 정책이 있는 역할을 생성합니다 AWS IoT Events.
2. AWS IoT Events 콘솔로 이동합니다.
3. 탐색 창에서 설정을 선택합니다.
4. 설정 페이지에서 편집을 선택합니다.
5. 로깅 옵션 편집 페이지의 로깅 옵션 섹션에서 다음을 수행합니다.
   a. 세부 정보 수준에서 옵션을 선택합니다.
   b. 역할 선택에서 선택한 로깅 작업을 수행할 수 있는 충분한 권한이 있는 역할을 선택합니다.
   c. (선택 사항) 세부 수준에 디버그를 선택한 경우 다음을 수행하여 디버그 대상을 추가할 수 있습니다.
      i. 대상 디버그에서 모델 옵션 추가를 선택합니다.
      ii. 감지기 모델 이름과 (선택 사항) KeyValue를 입력하여 로깅할 감지기 모델과 특정 감지기(인스턴스)를 지정합니다.
6. 업데이트를 선택합니다.

로깅 옵션이 성공적으로 업데이트되었습니다.

AWS IoT Events 콘솔에서 작업할 때 정기적으로 게시하여 감지기 모델을 저장합니다.

AWS IoT Events 콘솔을 사용하면 진행 중인 작업이 브라우저에 로컬로 저장됩니다. 하지만 감지기 모델을 AWS IoT Events에 저장하려면 게시를 선택해야 합니다. 감지기 모델을 게시하고 나면 계정에 액세스하는 데 사용하는 모든 브라우저에서 게시된 작업을 사용할 수 있게 됩니다.

> **Note**
>
> 작업을 게시하지 않으면 저장되지 않습니다. 감지기 모델을 게시한 후에는 모델 이름을 변경할 수 없습니다. 하지만 정의는 계속 수정할 수 있습니다.

AWS IoT Events 사용 사례에 대한 자습서

AWS IoT Events 자습서에서는 기본 설정 AWS IoT Events부터 보다 구체적인 사용 사례에 이르기까지의 다양한 측면을 다루는 절차 모음을 제공합니다. 각 자습서에서는 감지기 모델 생성, 입력 구성, 작업 설정, 다른 AWS 서비스와의 통합을 통해 강력한 IoT 솔루션을 만드는 데 도움이 되는 실제 기술을 구축하는 데 도움이 되는 실제 시나리오의 예를 보여줍니다.

이 장에서는 다음을 수행하는 방법을 설명합니다.

- 감지기 모델에 포함할 상태를 결정하는 데 도움을 받고, 감지기 인스턴스가 하나 혹은 여러 개가 필요할지 결정하십시오.
- AWS CLI를 사용하는 예를 따르십시오.
- 디바이스로부터 원격 측정 데이터를 수신하는 입력과 감지기 모델을 생성하여 해당 데이터를 전송하는 디바이스의 상태를 모니터링하고 보고할 수 있습니다.
- 입력, 감지기 모델 및 AWS IoT Events 서비스에 대한 제한과 한계를 검토하십시오.
- 감지기 모델의 더 복잡한 예시를 설명과 함께 참조하십시오.

주제

AWS IoT Events 를 사용하여 IoT 디바이스 모니터링

AWS IoT Events 를 사용하여 디바이스 또는 프로세스를 모니터링하고 중요한 이벤트에 따라 조치를 취할 수 있습니다. 이렇게 하려면 다음 기본 단계를 따르십시오.

입력 생성

디바이스와 프로세스가 원격 측정 데이터를 AWS IoT Events로 보낼 방법이 있어야 합니다. 이를 통해 메시지를 AWS IoT Events에 입력으로 전송합니다. 다음과 같은 여러 방법으로 메시지를 입력으로 보낼 수 있습니다.

- BatchPutMessage 작업을 사용합니다.

- `iotEvents` rule-action을 AWS IoT Core 규칙 엔진으로 정의합니다. rule-action은 입력의 메시지 데이터를 AWS IoT Events로 전달합니다.
- 에서 CreateDataset 작업을 AWS IoT 분석사용하여 로 데이터 세트를 생성합니다`contentDeliveryRules`. 이러한 규칙은 데이터 세트 콘텐츠가 자동으로 전송되는 AWS IoT Events 입력을 지정합니다.
- AWS IoT Events 감지기 모델의 `onInput` `onExit` 또는 `transitionEvents` 이벤트에서 iotEvents 작업을 정의합니다. 작업을 시작한 이벤트와 감지기 모델 인스턴스에 대한 정보가 지정된 이름을 사용하여 시스템에 입력으로 피드백됩니다.

디바이스가 이러한 방식으로 데이터를 전송하기 전에 하나 이상의 입력을 정의해야 합니다. 이렇게 하려면 각 입력에 이름을 지정하고 입력이 모니터링하는 수신 메시지 데이터의 필드를 지정합니다. 는 여러 소스에서 JSON 페이로드 형태로 입력을 AWS IoT Events 수신합니다. 각 입력을 자체적으로 처리하거나 다른 입력과 결합하여 더 복잡한 이벤트를 감지할 수 있습니다.

감지기 모델 생성

상태를 사용하여 감지기 모델(디바이스 또는 프로세스의 모델)을 정의합니다. 각 상태에 대해 수신되는 입력을 평가하여 중요 이벤트를 탐지하는 조건부(부울) 논리를 정의합니다. 이벤트가 감지되면 상태를 변경하거나 다른 AWS 서비스를 사용하여 사용자 지정 빌드 또는 사전 정의된 작업을 시작할 수 있습니다. 특정 상태가 시작 또는 종료할 때 또한 선택적으로 특정 조건이 충족될 때 작업을 시작하는 추가 이벤트를 정의할 수 있습니다.

이 자습서에서는 모델이 특정 상태에 들어가거나 종료될 때 Amazon SNS 메시지를 작업으로 전송합니다.

디바이스 또는 프로세스 모니터링

여러 디바이스 또는 프로세스를 모니터링하는 경우 각 입력에 특정 디바이스를 식별하거나 입력의 출처를 처리하는 필드를 지정합니다. (`key`의 `CreateDetectorModel` 필드 참조.) 새 장치가 식별되면(`key`로 식별되는 입력 필드에 새 값이 표시됨) 감지기가 생성됩니다. (각 감지기는 감지기 모델의 인스턴스입니다.) 그러면 새 감지기는 해당 감지기 모델이 업데이트되거나 삭제될 때까지 해당 디바이스에서 오는 입력에 계속 응답합니다.

단일 프로세스를 모니터링하는 경우 여러 디바이스 또는 하위 프로세스가 입력을 보낸다 하더라도 고유한 식별 `key` 필드를 지정하지 마십시오. 이 경우 첫 번째 입력이 도착하면 단일 감지기(인스턴스)가 생성됩니다.

메시지를 감지기 모델에 입력으로 전송

디바이스에서 메시지를 보내거나 AWS IoT Events 감지기에 대한 입력으로 처리하는 방법에는 여러 가지가 있습니다.이 경우 메시지에 대한 추가 형식을 지정할 필요가 없습니다. 이 자습서에서는

AWS IoT 콘솔을 사용하여 메시지 데이터를 전달하는 AWS IoT Core 규칙 엔진에 대한 AWS IoT Events 작업 규칙을 작성합니다 AWS IoT Events. 이렇게 하려면 입력을 이름으로 식별해야 합니다. 그런 다음 AWS IoT 콘솔을 계속 사용하여 입력으로 전달되는 일부 메시지를 생성합니다 AWS IoT Events.

감지기 모델에 어떤 상태가 필요한지 어떻게 알 수 있나요?

감지기 모델의 상태를 결정하려면 취할 수 있는 조치를 먼저 결정하십시오. 예를 들어, 자동차가 휘발유를 사용하는 경우 여행을 시작할 때 연료 게이지를 보고 연료 충전이 필요한지 확인합니다. 여기서 한 가지 조치를 취할 수 있습니다. 운전자에게 "연료를 충전하십시오"라고 말하는 것입니다. 감지기 모델에는 "자동차에 연료가 필요하지 않음"과 "자동차에 연료가 필요함"이라는 두 가지 상태가 필요합니다. 일반적으로 가능한 각 동작에 대해 하나의 상태를 정의하고, 조치가 필요하지 않은 경우를 위한 상태를 하나 더 정의하려고 합니다. 이 방법은 작업 자체가 더 복잡한 경우에도 적용됩니다. 예를 들어 가장 가까운 주유소 또는 가장 저렴한 주유소의 위치 정보를 찾아보고 포함할 수 있지만, 이는 "연료를 충전하십시오"라는 메시지를 보낼 때 이렇게 합니다.

다음으로 전환할 상태를 결정하려면 입력 내용을 살펴보십시오. 입력에는 어떤 상태로 전환되어야 하는지 결정하는 데 필요한 정보가 들어 있습니다. 입력을 만들려면 디바이스나 프로세스에서 보낸 메시지에서 결정을 내리는 데 도움이 되는 필드를 하나 이상 선택합니다. 이 예시에서는 현재 연료 수준("연료 비율")을 알려주는 입력이 하나 필요합니다. 자동차가 여러 개의 서로 다른 필드를 포함하는 서로 다른 메시지를 여러 개 보내고 있을 수 있습니다. 이 입력을 생성하려면 메시지와 현재 연료 게이지 수준을 보고하는 필드를 선택해야 합니다. 이동하려는 거리("목적지까지의 거리")를 하드코딩하여 단순하게 만들 수 있습니다. 즉, 평균 이동 거리를 사용할 수 있습니다. 입력값을 바탕으로 계산을 할 수 있습니다(연료 비율이 몇 갤런으로 환산되나요? 보유한 갤런과 평균 "갤런당 마일"을 감안하면 평균 이동 거리가 주행 가능 거리보다 길까요?). 이러한 계산을 수행하고 이벤트에서 메시지를 전송합니다.

지금까지는 두 개의 상태와 한 개의 입력이 있습니다. 첫 번째 상태에는 입력을 기반으로 계산을 수행하고 두 번째 상태로 이동 여부를 결정하는 이벤트가 필요합니다. 이는 전환 이벤트입니다. (`transitionEvents`가 상태의 `onInput` 이벤트 목록에 있습니다. 첫 번째 상태에서 입력을 받은 후, 이벤트의 `condition`이 충족되면 이벤트가 두 번째 상태로 전환됩니다.) 두 번째 상태에 도달하면 해당 상태로 전환되는 즉시 메시지가 전송됩니다. (`onEnter` 이벤트를 사용합니다. 두 번째 상태로 전환되면 이 이벤트가 메시지를 보냅니다. 다른 입력이 도착할 때까지 기다릴 필요가 없습니다.) 다른 유형의 이벤트도 있지만 여기까지가 간단한 예시입니다.

다른 유형의 이벤트는 `onExit` 및 `onInput`입니다. 입력이 수신되고 조건이 충족되는 즉시 `onInput` 이벤트는 지정된 작업을 수행합니다. 작업이 현재 상태를 종료하고 조건이 충족되면 `onExit` 이벤트는 지정된 작업을 수행합니다.

놓친 것이 있나요? 네, 어떻게 하면 첫 번째 상태 “자동차에 연료가 필요하지 않음”으로 돌아갈 수 있을까요? 연료 탱크를 가득 채우면 입력란에 탱크가 가득 찼다고 표시됩니다. 두 번째 상태에서는 (두 번째 상태의 `onInput:` 이벤트에서) 입력이 수신될 때 발생하는 첫 번째 상태로의 전환 이벤트가 필요합니다. 원하는 곳으로 갈 수 있을 만큼 연료가 충분하다는 계산이 나오면 다시 첫 번째 상태로 전환해야 합니다.

이것이 기본입니다. 일부 감지기 모델은 가능한 동작뿐만 아니라 중요한 입력을 반영하는 상태를 추가하여 더 복잡해집니다. 예를 들어 온도를 추적하는 감지기 모델에는 “정상”, “고온”, “잠재적 문제”의 세 가지 상태가 있을 수 있습니다. 온도가 특정 수준 이상으로 올라갔지만 아직 너무 뜨거워지지 않은 경우 잠재적 문제 상태로 전환됩니다. 이 온도로 15분 이상 유지되지 않으면 경보를 보내고 싶지 않을 것입니다. 그 전에 온도가 정상으로 돌아오면 감지기는 정상 상태로 다시 전환됩니다. 타이머가 만료되면 감지기가 너무 뜨거워진 상태로 전환되어 경보를 보내므로 주의해야 합니다. 변수와 더 복잡한 이벤트 조건 세트를 사용하여 동일한 작업을 수행할 수 있습니다. 하지만 실제로는 다른 상태를 사용하여 계산 결과를 저장하는 것이 더 쉬운 경우가 많습니다.

감지기 인스턴스가 하나 필요한지 아니면 여러 개가 필요한지 어떻게 알 수 있습니까?

필요한 인스턴스 수를 결정하려면 “무엇을 알고 싶은가요?”라고 스스로 질문하십시오. 오늘 날씨가 어떤지 알고 싶다고 가정해 봅시다. 비가 오나요(상태)? 우산을 써야 하나요(작업)? 온도를 알려주는 센서, 습도를 알려주는 센서, 기압, 풍속, 강수를 알려주는 센서를 사용할 수 있습니다. 하지만 이러한 모든 센서를 함께 모니터링하여 날씨 상태(비, 눈, 흐린 날씨, 맑음)와 적절한 작업(우산 챙기기 또는 자외선 차단제 바르기)을 파악해야 합니다. 센서 수가 많더라도 하나의 감지기 인스턴스가 날씨 상태를 모니터링하고 취해야 할 작업을 알려주는 것이 좋습니다.

하지만 해당 리전의 일기 예보관이라면 리전 곳곳의 다양한 위치에 이러한 센서 그룹이 여러 개 있을 수 있습니다. 각 지역의 사람들은 해당 지역의 날씨가 어떤지 알아야 합니다. 이 경우 여러 개의 감지기 인스턴스가 필요합니다. 각 위치의 각 센서가 보고하는 데이터에는 `key` 필드로 지정한 필드가 포함되어야 합니다. 이 필드를 사용하면 AWS IoT Events 가 해당 영역에 대한 감지기 인스턴스를 만든 다음, 해당 감지기 인스턴스가 도착할 때마다 이 정보를 해당 감지기 인스턴스로 계속 라우팅할 수 있습니다. 날씨 때문에 머리 스타일을 망치거나 얼굴이 타는 일은 더 이상 없습니다!

기본적으로 모니터링해야 할 한 개의 상황(프로세스 또는 위치 한 개)이 있다면 한 개의 감지기 인스턴스가 필요합니다. 모니터링해야 할 상황(위치, 프로세스)이 많은 경우 여러 개의 감지기 인스턴스가 필요합니다.

CLI를 사용하여 두 상태에 대한 AWS IoT Events 감지기 생성

이 예제에서는 AWS CLI 명령을 사용하여 AWS IoT Events APIs를 호출하여 엔진의 두 가지 상태인 정상 상태와 과압 조건을 모델링하는 감지기를 생성합니다.

엔진에서 측정된 압력이 특정 임계값을 초과하면 모델은 과압 상태로 전환하고 Amazon Simple Notification Service(SNS) 메시지를 보내 기술자에게 상태를 알립니다. 압력 판독값이 3회 연속 임계값 아래로 떨어지면 모델은 정상 상태로 돌아가고 상태가 해결되었음을 확인하는 Amazon SNS 메시지를 하나 더 보냅니다. 비선형 복구 단계 또는 일회성 변칙 복구 측정값 발생 시 과압/정상 메시지가 끊길 가능성을 없애려면 압력 판독값이 임계값 아래로 3회 연속 측정된 값이 필요합니다.

다음은 감지기 생성 단계의 개요입니다.

입력을 생성합니다.

디바이스와 프로세스를 모니터링하려면 원격 측정 데이터를 AWS IoT Events로 보낼 방법이 있어야 합니다. 이는 메시지를 입력으로 전송하여 수행됩니다 AWS IoT Events. 여러 가지 방법으로 이 작업을 수행할 수 있습니다.

- BatchPutMessage 작업을 사용합니다. 이 방법은 쉽지만 디바이스 또는 프로세스가 SDK 또는를 통해 AWS IoT Events API에 액세스할 수 있어야 합니다 AWS CLI.
- 에서 메시지 데이터를 전달하는 AWS IoT Core 규칙 엔진에 대한 AWS IoT Events 작업 규칙을 AWS IoT Core작성합니다 AWS IoT Events. 이렇게 하면 입력이 이름으로 식별됩니다. 디바이스 또는 프로세스가 메시지를 전송할 수 있거나 이미 전송 중인 경우이 방법을 사용합니다 AWS IoT Core. 이 방법을 사용하면 일반적으로 장치의 컴퓨팅 성능이 덜 필요합니다.
- 에서 CreateDataset 작업을 AWS IoT Analytics사용하여 데이터 세트 콘텐츠가 자동으로 전송되는 AWS IoT Events 입력을 `contentDeliveryRules` 지정하는를 사용하여 데이터 세트를 생성합니다. AWS IoT 분석에서 집계되거나 분석된 데이터를 기반으로 디바이스 또는 프로세스를 제어하려면 이 방법을 사용하십시오.

디바이스에서 이러한 방식으로 데이터를 전송하려면 먼저 하나 이상의 입력을 정의해야 합니다. 이렇게 하려면 각 입력에 이름을 지정하고 입력이 모니터링하는 수신 메시지 데이터의 필드를 지정하십시오.

감지기 모델 생성

상태를 사용하여 감지기 모델(장비 또는 프로세스의 모델)을 생성합니다. 각 상태에 대해 수신되는 입력을 평가하여 중요 이벤트를 탐지하는 조건부(부울) 논리를 정의합니다. 이벤트가 감지되면 상태를 변경하거나 다른 AWS 서비스를 사용하여 사용자 지정 빌드 또는 사전 정의된 작업을 시작할

수 있습니다. 특정 상태가 시작 또는 종료할 때 또한 선택적으로 특정 조건이 충족될 때 작업을 시작하는 추가 이벤트를 정의할 수 있습니다.

여러 디바이스 또는 프로세스 모니터링

여러 디바이스 또는 프로세스를 모니터링하고 각 디바이스를 개별적으로 추적하려면 각 입력에 특정 디바이스를 식별하거나 입력이 들어오는 프로세스를 식별하는 필드를 지정하십시오. `CreateDetectorModel`의 `key` 필드를 참조하십시오. 새 디바이스가 식별되면 (`key`로 식별되는 입력 필드에 새 값이 표시됨) 감지기 인스턴스가 생성됩니다. 새 감지기 인스턴스는 해당 감지기 모델이 업데이트되거나 삭제될 때까지 해당 특정 디바이스에서 들어오는 입력에 계속 응답합니다. 입력 `key` 필드의 고유 값 수만큼 고유한 감지기(인스턴스)가 있습니다.

단일 디바이스 또는 프로세스 모니터링

단일 프로세스를 모니터링하는 경우 여러 디바이스 또는 하위 프로세스가 입력을 보낸다 하더라도 고유한 식별 `key` 필드를 지정하지 마십시오. 이 경우 첫 번째 입력이 도착하면 단일 감지기(인스턴스)가 생성됩니다. 예를 들어 집 내부의 각 방에는 온도 센서가 있지만 집 전체를 냉난방하는 HVAC 장치는 하나뿐일 수 있습니다. 따라서 각 방의 사용자가 자신의 의견(입력)이 수용되기를 원하더라도 이를 단일 프로세스로만 제어할 수 있습니다.

디바이스 또는 프로세스의 메시지를 감지기 모델에 입력으로 전송

디바이스 또는 프로세스에서 입력의 AWS IoT Events 감지기에 대한 입력으로 메시지를 보내는 몇 가지 방법을 설명했습니다. 입력을 생성하고 감지기 모델을 구축했으면 이제 데이터 전송을 시작할 준비가 되었습니다.

> **Note**
>
> 감지기 모델을 생성하거나 기존 감지기 모델을 업데이트하면 신규 또는 업데이트된 감지기 모델이 메시지를 수신하고 감지기(인스턴스) 생성을 시작하기까지 몇 분이 걸립니다. 감지기 모델이 업데이트되면 이 기간 동안 이전 버전에 기반한 동작이 계속 나타날 수 있습니다.

주제

- AWS IoT Events 입력을 생성하여 디바이스 데이터 캡처
- 에서 디바이스 상태를 나타내는 감지기 모델 생성 AWS IoT Events
- 에서 감지기에 입력으로 메시지 전송 AWS IoT Events

AWS IoT Events 입력을 생성하여 디바이스 데이터 캡처

에 대한 입력을 설정할 때를 활용하여 디바이스가 센서 데이터를 통신 AWS CLI 하는 방법을 정의할 AWS IoT Events수 있습니다. 예를 들어 디바이스가 모터 식별자 및 센서 판독값과 함께 JSON 형식의 메시지를 전송하는 경우 압력 및 모터 ID와 같은 메시지의 특정 속성을 매핑하는 입력을 생성하여이 데이터를 캡처할 수 있습니다. 프로세스는 JSON 파일에서 입력을 정의하고, 관련 데이터 포인트를 지정하고,를 사용하여 입력을 등록하는 AWS CLI 것으로 시작됩니다 AWS IoT Events. 이를 통해 AWS IoT는 실시간 센서 데이터를 기반으로 중요한 조건을 모니터링하고 이에 대응할 수 있습니다.

예를 들어 다음 형식으로 메시지를 전송한다고 가정해 보겠습니다.

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

다음 AWS CLI 명령을 사용하여 `pressure` 데이터와 `motorid` (메시지를 전송한 특정 디바이스를 식별하는)를 캡처하는 입력을 생성할 수 있습니다.

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

파일 `pressureInput.json`에는 다음이 포함되어 있습니다.

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

직접 입력을 생성할 때는 먼저 디바이스 또는 프로세스에서 예제 메시지를 JSON 파일로 수집해야 합니다. 이를 사용해 콘솔 또는 CLI에서 입력을 생성할 수 있습니다.

에서 디바이스 상태를 나타내는 감지기 모델 생성 AWS IoT Events

AWS IoT Events 입력을 생성하여 디바이스 데이터 캡처에서 모터의 압력 데이터를 보고하는 메시지를 기반으로 input을 만들었습니다. 예제를 계속하기 위해 모터의 과압 상황에 대응하는 감지기 모델을 살펴보겠습니다.

“Normal” 및 “Dangerous”의 두 가지 상태를 생성합니다. 각 감지기(인스턴스)는 생성될 때 "Normal" 상태로 전환됩니다. 인스턴스는 key “motorid”에 대한 고유한 값을 가진 입력이 도착하면 생성됩니다.

감지기 인스턴스가 70 이상의 압력 판독값을 수신하면 "Dangerous" 상태로 전환되고 Amazon SNS 메시지를 경고로 보냅니다. 압력 측정값이 정상(70 미만)으로 3회 연속 입력되면 감지기는 "Normal" 상태로 돌아가고 다른 Amazon SNS 메시지를 모두 지우기 상태로 보냅니다.

이 예제 감지기 모델은 정의에 Amazon 리소스 이름(ARN)이 "targetArn": "arn:aws:sns:us-east-1:123456789012:underPressureAction" 및 "targetArn": "arn:aws:sns:us-east-1:123456789012:pressureClearedAction"으로 표시된 두 개의 Amazon SNS 주제를 생성했다고 가정합니다.

자세한 내용은 Amazon Simple Notification Service 개발자 안내서를 참조하십시오. 더 구체적인 설명은 Amazon Simple Notification Service API 참조의 CreateTopic 작업을 참조하십시오.

또한이 예제에서는 적절한 권한이 있는 AWS Identity and Access Management (IAM) 역할을 생성했다고 가정합니다. 이 역할의 ARN은 감지기 모델 정의에 "roleArn": "arn:aws:iam::123456789012:role/IoTEventsRole"과 같이 표시됩니다. 에 대한 권한 설정 AWS IoT Events의 단계에 따라 이 역할을 생성하고 역할의 ARN을 감지기 모델 정의의 적절한 위치에 복사하십시오.

다음 AWS CLI 명령을 사용하여 감지기 모델을 생성할 수 있습니다.

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

"motorDetectorModel.json" 파일에는 다음 코드가 포함되어 있습니다.

```
{
  "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"
}
```

에서 감지기에 입력으로 메시지 전송 AWS IoT Events

이제 디바이스에서 보낸 메시지의 중요 필드를 식별하는 입력이 정의되었습니다(AWS IoT Events 입력을 생성하여 디바이스 데이터 캡처 참조). 이전 섹션에서는 모터의 과압 이벤트에 반응하는 `detector model`을 생성했습니다(에서 디바이스 상태를 나타내는 감지기 모델 생성 AWS IoT Events 참조).

예제를 완료하려면 디바이스(이 경우에는 AWS CLI 가 설치된 컴퓨터)의 메시지를 감지기에 입력으로 보내십시오.

> **Note**
>
> 감지기 모델을 생성하거나 기존 감지기 모델을 업데이트하면 새 또는 업데이트된 감지기 모델이 메시지를 수신하고 감지기(인스턴스)를 생성하기 시작하기까지 몇 분 정도 걸립니다. 감지기 모델을 업데이트하면 이 기간 동안 이전 버전에 기반한 동작이 계속 나타날 수 있습니다.

다음 AWS CLI 명령을 사용하여 임계값을 위반하는 데이터가 포함된 메시지를 보냅니다.

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

파일 “highPressureMessage.json”에는 다음이 포함되어 있습니다.

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

전송되는 각 메시지에서 messageId를 변경해야 합니다. 변경하지 않으면 AWS IoT Events 시스템은 메시지를 중복 제거합니다. 메시지는 지난 5분 이내에 전송된 다른 메시지messageID와 동일한 경우 AWS IoT Events 무시합니다.

이때 모터 "Fulton-A32"의 이벤트를 모니터링하기 위한 감지기(인스턴스)가 생성됩니다. 이 감지기는 생성되었을 때의 "Normal" 상태로 들어갑니다. 하지만 임계값을 초과하는 압력 값을 보냈기 때문에 즉시 "Dangerous" 상태로 전환됩니다. 그러면 감지기는 ARN이 arn:aws:sns:us-east-1:123456789012:underPressureAction인 Amazon SNS 엔드포인트로 메시지를 전송합니다.

다음 AWS CLI 명령을 실행하여 압력 임계값 미만의 데이터가 포함된 메시지를 보냅니다.

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

파일 normalPressureMessage.json에는 다음이 포함되어 있습니다.

```
{
  "messages": [
    {
      "messageId": "00002",
      "inputName": "PressureInput",
```

```
      "payload": "{\"motorid\": \"Fulton-A32\", \"sensorData\": {\"pressure\": 60,
 \"temperature\": 29} }"
    }
  ]
}
```

5분 이내에 BatchPutMessage 명령을 호출할 때마다 파일의 messageId를 변경해야 합니다. 메시지를 두 번 더 보내십시오. 메시지가 세 번 전송되면 모터 “Fulton-A32”의 감지기(인스턴스)가 Amazon SNS 엔드포인트 "arn:aws:sns:us-east-1:123456789012:pressureClearedAction"으로 메시지를 보내고 "Normal" 상태를 다시 입력합니다.

> **Note**
>
> BatchPutMessage를 사용하여 한 번에 여러 메시지를 보낼 수 있습니다. 하지만 이러한 메시지가 처리되는 순서는 보장되지 않습니다. 메시지(입력)가 순서대로 처리되도록 하려면 메시지를 한 번에 하나씩 보내고 API가 호출될 때마다 응답 성공을 기다리십시오.

다음은 이 섹션에 설명된 감지기 모델 예제로 생성된 SNS 메시지 페이로드의 예시입니다.

“압력 임계값 위반” 이벤트 발생 시

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

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

“정상 압력 복원” 이벤트 발생 시

```
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"
}
```

타이머를 정의한 경우 해당 타이머의 현재 상태는 SNS 메시지 페이로드에도 표시됩니다.

메시지 페이로드에는 메시지가 전송된 시점 (즉, SNS 작업이 실행된 시점)의 감지기(인스턴스) 상태에 대한 정보가 포함됩니다. https://docs.aws.amazon.com/iotevents/latest/apireference/API_iotevents-data_DescribeDetector.html 작업을 사용하여 감지기 상태에 대한 유사한 정보를 얻을 수 있습니다.

AWS IoT Events 감지기 모델 제한 사항 및 제한 사항

감지기 모델을 생성할 때 고려해야 할 중요한 사항은 다음과 같습니다.

actions 필드 사용 방법

actions 값은 객체의 목록입니다. 객체를 두 개 이상 가질 수 있지만 각 객체에는 하나의 작업만 허용됩니다.

Example

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

condition 필드 사용 방법

condition은(는) transitionEvents에 필수 항목이며 다른 경우에는 선택 사항입니다.

해당 condition 필드가 없는 경우 이는 "condition": true와 같습니다.

조건 표현식의 평가 결과는 부울 값이어야 합니다. 결과가 부울 값이 아닌 경우 해당 값은 false와 동일하며 이벤트에서 지정된 nextState로의 전환 또는 actions으로 시작되지 않습니다.

변수 값의 사용 가능 여부

기본적으로 변수 값이 이벤트에 설정된 경우 새 값을 사용할 수 없거나 같은 그룹 내 다른 이벤트의 조건을 평가하는 데 사용할 수 없습니다. 새 값은 사용할 수 없거나 동일한 onInput, onEnter 또는 onExit 필드의 이벤트 조건에 사용되었습니다.

감지기 모델 정의에서 evaluationMethod 파라미터를 설정하여 이 동작을 변경하십시오. evaluationMethod를 SERIAL로 설정하면 변수가 업데이트되고 이벤트가 정의된 순서대로 이벤트 조건이 평가됩니다. 그렇지 않고 evaluationMethod를 BATCH로 설정하거나 기본값으로 설정하면, 상태 내의 변수가 업데이트되고 모든 이벤트 조건이 평가된 후에만 상태 내의 이벤트가 수행됩니다.

"Dangerous" 상태, onInput 필드에서 조건이 충족되는 경우(현재 입력의 압력이 70 이하인 경우), "Pressure Okay" 이벤트의 "$variable.pressureThresholdBreached"가 하나씩 감소합니다.

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

감지기는 "$variable.pressureThresholdBreached"가 0에 도달했을 때 (즉, 감지기가 70 이하의 압력 측정값을 연속으로 세 번 수신했을 때) "Normal" 상태로 다시 전환되어야 합니다. transitionEvents의 "BackToNormal" 이벤트는 "$variable.pressureThresholdBreached"가 1보다 작거나 같은지(0이 아님) 테스트하고 "$input.PressureInput.sensorData.pressure"에서 제공한 현재 값이 70 미만인지 다시 확인해야 합니다.

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

그렇지 않으면 조건에서 변수 값만 테스트할 경우 정상 판독값 2개와 과압 판독값이 뒤따르는 경우 조건을 충족하고 "Normal" 상태로 다시 전환됩니다. 조건은 이전에 입력이 처리되었을 때 "$variable.pressureThresholdBreached"에 제공된 값을 확인하는 것입니다. 이 경우 "Overpressurized" 이벤트에서 변수 값이 3으로 재설정되지만 이 새 값은 어떤 condition도 사용할 수 없다는 점에 유의하십시오.

기본적으로 컨트롤이 onInput 필드에 들어갈 때마다 condition은(는) onInput에서 지정한 작업에 의해 변수가 변경되기 전의 입력 처리 시작 시점의 변수 값만 볼 수 있습니다. onEnter 및 onExit도 마찬가지입니다. 상태를 입력하거나 종료할 때 변수에 적용된 변경 사항은 동일한 onEnter 또는 onExit 필드에 지정된 다른 조건에서는 사용할 수 없습니다.

감지기 모델 업데이트 시 지연 시간

감지기 모델을 업데이트, 삭제 및 재생성하는 경우(UpdateDetectorModel 참조), 생성된 모든 감지기(인스턴스)가 삭제되고 새 모델을 사용하여 감지기를 다시 생성하기까지 약간의 지연이 발생합니다. 새 감지기 모델이 적용되고 새 입력이 도착한 후에 다시 생성됩니다. 이 기간 동안에는 이전 버전의 감지기 모델에서 생성된 감지기가 입력을 계속 처리할 수 있습니다. 이 기간 동안에는 이전 감지기 모델에서 정의한 알림을 계속 받을 수 있습니다.

입력 키의 공백

입력 키에는 공백이 허용되지만 키 참조는 입력 속성 정의와 표현식에서 키 값을 참조할 때 모두 백틱으로 묶어야 합니다. 예를 들어 다음과 같은 메시지 페이로드가 있다고 가정합니다.

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

다음을 사용하여 입력을 정의합니다.

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

조건 표현식에서는 백틱도 사용하여 이러한 키의 값을 참조해야 합니다.

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

주석이 달린 예제:를 사용한 HVAC 온도 제어 AWS IoT Events

다음 예제 JSON 파일 중 일부에는 주석이 인라인으로 포함되어 있어 유효하지 않은 JSON이 됩니다. 이러한 예제의 전체 버전(주석 제외)은 예:에서 HVAC 온도 제어 사용 AWS IoT Events에서 확인할 수 있습니다.

이 예제는 다음 작업을 수행할 수 있는 온도 조절기 제어 모델을 구현합니다.

- 여러 영역을 모니터링하고 제어하는 데 사용할 수 있는 감지기 모델을 하나만 정의합니다. 감지기 인스턴스는 각 영역에 대해 생성됩니다.
- 각 제어 영역의 여러 센서에서 온도 데이터를 수집합니다.
- 특정 지역의 온도 설정점을 변경합니다.
- 인스턴스가 사용 중인 동안 각 영역의 작동 파라미터를 설정하고 이러한 파라미터를 재설정합니다.
- 영역에 센서를 동적으로 추가 또는 삭제합니다.
- 냉난방 장치를 보호할 수 있는 최소 런타임을 지정하십시오.
- 비정상적인 센서 판독값을 거부합니다.
- 센서 중 하나에서 지정된 임계값보다 온도가 높거나 낮다고 보고하는 경우 즉시 난방 또는 냉방을 작동시키는 비상 설정점을 정의합니다.
- 비정상적인 측정값 및 온도 급상승을 보고합니다.

주제

- 의 감지기 모델에 대한 입력 정의 AWS IoT Events
- AWS IoT Events 감지기 모델 정의 생성
- BatchUpdateDetector를 사용하여 AWS IoT Events 감지기 모델 업데이트
- 의 입력에 BatchPutMessage 사용 AWS IoT Events
- 에서 MQTT 메시지 수집 AWS IoT Events
- 에서 Amazon SNS 메시지 생성 AWS IoT Events
- 에서 DescribeDetector API 구성 AWS IoT Events
- 에 규칙 AWS IoT Core 엔진 사용 AWS IoT Events

의 감지기 모델에 대한 입력 정의 AWS IoT Events

여러 영역의 온도를 모니터링하고 제어하는 데 사용할 수 있는 하나의 감지기 모델을 만들려고 합니다. 각 영역에는 온도를 보고하는 센서가 여러 개 있을 수 있습니다. 각 구역에는 냉난방 장치가 각각 하나

씩 있으며, 이 장치를 켜거나 끄면 해당 구역의 온도를 제어할 수 있다고 가정합니다. 각 영역은 하나의 감지기 인스턴스로 제어됩니다.

모니터링하고 제어하는 영역마다 특성이 다르기 때문에 제어 파라미터가 다를 수 있으므로 각 영역에 해당 파라미터를 제공하도록 `'seedTemperatureInput'`을 정의합니다. 이러한 입력 메시지 중 하나를 AWS IoT Events에 보내면 해당 영역에서 사용하려는 파라미터가 포함된 새 감지기 모델 인스턴스가 생성됩니다. 해당 입력의 정의는 다음과 같습니다.

CLI 명령:

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

파일: `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" }
    ]
  }
}
```

응답:

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

참고

- 모든 메시지에서 'areaId' 수신된 각 고유자에 대해 새 감지기 인스턴스가 생성됩니다. 'areaDetectorModel' 정의의 'key' 필드를 참조하십시오.
- 평균 온도는 'allowedError'에 의해 해당 지역의 난방 또는 냉방 장치가 활성화되기 전의 'desiredTemperature'와 다를 수 있습니다.
- 센서가 'rangeHigh'보다 높은 온도를 보고하면 감지기가 급상승을 보고하고 즉시 냉각 장치를 가동합니다.
- 센서가 'rangeLow'보다 낮은 온도를 보고하면 감지기는 스파이크를 보고하고 즉시 난방 장치를 가동합니다.
- 센서가 'anomalousHigh'보다 높거나 'anomalousLow'보다 낮은 온도를 보고하는 경우 감지기는 비정상적인 센서 판독값을 보고하지만 보고된 온도 판독값은 무시합니다.
- 'sensorCount'은(는) 해당 구역에 대해 보고하는 센서 수를 감지기에 알려줍니다. 감지기는 수신한 각 온도 측정값에 적절한 가중치를 부여하여 해당 구역의 평균 온도를 계산합니다. 따라서 감지기는 각 센서가 보고하는 내용을 추적할 필요가 없으며 필요에 따라 센서 수를 동적으로 변경할 수 있습니다. 하지만 개별 센서가 오프라인 상태가 되면 감지기는 이를 인식하지 못하거나 허용치를 적용하지 못합니다. 각 센서의 연결 상태를 모니터링하기 위해 다른 감지기 모델을 특별히 생성하는 것이 좋습니다. 상호 보완적인 두 개의 감지기 모델을 사용하면 두 모델의 설계가 단순해집니다.
- 'noDelay' 값은 true 또는 false일 수 있습니다. 장치의 무결성을 보호하고 작동 수명을 연장하려면 난방 또는 냉방 장치를 작동한 후, 최소 일정 시간 동안 켜두어야 합니다. 'noDelay'가 false로 설정된 경우, 감지기 인스턴스는 냉방 및 난방 장치의 전원을 끄기 전에 지연 시간을 적용하여 최소 시간 동안 작동되도록 합니다. 변수 값을 사용하여 타이머를 설정할 수 없기 때문에 감지기 모델 정의에 지연 시간(초)이 하드코딩되었습니다.

센서 데이터를 감지기 인스턴스로 전송하는 데 'temperatureInput'이 사용됩니다.

CLI 명령:

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

파일: temperatureInput.json

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

응답:

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

참고

- 예제 감지기 인스턴스에서 센서를 직접 제어하거나 모니터링하는 데 `'sensorId'`가 사용되지 않습니다. 감지기 인스턴스가 보낸 알림에 자동으로 전달됩니다. 이를 통해 오류가 발생한 센서(예: 비정상적인 판독값을 정기적으로 전송하는 센서가 곧 고장날 수 있음) 또는 오프라인 상태가 된 센서(장치의 심장 박동을 모니터링하는 추가 감지기 모델의 입력으로 사용되는 경우)를 식별하는 데 사용할 수 있습니다. 또한 측정값이 평균과 정기적으로 다를 경우 `'sensorId'`은(는) 해당 구역의 따뜻한 곳이나 차가운 곳을 식별하는 데도 도움이 됩니다.
- 센서의 데이터를 적절한 감지기 인스턴스로 라우팅하는 데 `'areaId'`가 사용됩니다. 감지기 인스턴스는 모든 메시지에서 `'areaId'` 수신된 각 고유에 대해 생성됩니다. `'areaDetectorModel'` 정의의 `'key'` 필드를 참조하십시오.

AWS IoT Events 감지기 모델 정의 생성

`'areaDetectorModel'` 예제에는 주석이 인라인으로 되어 있습니다.

CLI 명령:

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

파일: areaDetectorModel.json

```
{
  "detectorModelName": "areaDetectorModel",
  "detectorModelDefinition": {
    "states": [
      {
        "stateName": "start",
        // In the 'start' state we set up the operation parameters of the new detector
 instance.
        //   We get here when the first input message arrives. If that is a
 'seedTemperatureInput'
        //   message, we save the operation parameters, then transition to the 'idle'
 state. If
        //   the first message is a 'temperatureInput', we wait here until we get a
        //   'seedTemperatureInput' input to ensure our operation parameters are set.
 We can
        //   also reenter this state using the 'BatchUpdateDetector' API. This enables
 us to
        //   reset the operation parameters without needing to delete the detector
 instance.
        "onEnter": {
          "events": [
            {
              "eventName": "prepare",
              "condition": "true",
              "actions": [
                {
                  "setVariable": {
                    // initialize 'sensorId' to an invalid value (0) until an actual
 sensor reading
                    //   arrives
                    "variableName": "sensorId",
                    "value": "0"
                  }
                },
                {
                  "setVariable": {
```

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

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

```
            {
              "eventName": "reset",
              "condition": "($variable.resetMe == true) &&
($input.temperatureInput.sensorData.temperature < $variable.anomalousHigh &&
$input.temperatureInput.sensorData.temperature > $variable.anomalousLow)",
              // This event is triggered if we have reentered the 'start' state using
the
              //   'BatchUpdateDetector' API with 'resetMe' set to true. When we
reenter using
              //   'BatchUpdateDetector' we do not automatically continue to the 'idle'
state, but
              //   wait in 'start' until the next input message arrives. This event
enables us to
              //   transition to 'idle' on the next valid 'temperatureInput' message
that arrives.
              "actions": [
                {
                  "setVariable": {
                    "variableName": "averageTemperature",
                    "value": "((($variable.averageTemperature * ($variable.sensorCount
- 1)) + $input.temperatureInput.sensorData.temperature) / $variable.sensorCount)"
                  }
                }
              ],
              "nextState": "idle"
            }
          ]
        },
        "onExit": {
          "events": [
            {
              "eventName": "resetHeatCool",
              "condition": "true",
              // Make sure the heating and cooling units are off before entering
'idle'.
              "actions": [
                {
                  "sns": {
                    "targetArn": "arn:aws:sns:us-west-2:123456789012:heatOff"
                  }
                },
                {
                  "sns": {
                    "targetArn": "arn:aws:sns:us-west-2:123456789012:coolOff"
```

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


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

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

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

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

```
                    //   of a cooling/heating unit, but we only want to set the timer
when we
                    //   enter that new state initially.
                    "variableName": "enteringNewState",
                    "value": "true"
                  }
                }
              ],
              "nextState": "cooling"
            },

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

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

          {
            "eventName": "lowTemperatureThreshold",
            "condition": "((((($variable.averageTemperature * ($variable.sensorCount
- 1)) + $input.temperatureInput.sensorData.temperature) / $variable.sensorCount) <
($variable.desiredTemperature - $variable.allowedError))",
            // When the average temperature is below the desired temperature minus
the allowed error factor,
            //   it is time to start heating. Note that we calculate the average
temperature here again
```

```
              //   because the value stored in the 'averageTemperature' variable is not
yet available for use
              //   in our condition.
              "actions": [
                {
                  "sns": {
                    "targetArn": "arn:aws:sns:us-west-2:123456789012:heatOn"
                  }
                },
                {
                  "iotTopicPublish": {
                    "mqttTopic": "hvac/Heating/On"
                  }
                },
                {
                  "setVariable": {
                    "variableName": "enteringNewState",
                    "value": "true"
                  }
                }
              ],
              "nextState": "heating"
            }
          ]
        }
      },


      {
        "stateName": "cooling",
        "onEnter": {
          "events": [
            {
              "eventName": "delay",
              "condition": "!$variable.noDelay && $variable.enteringNewState",
              // If the operational parameters specify that there should be a minimum
time that the
              //   heating and cooling units should be run before being shut off again,
we set
              //   a timer to ensure the proper operation here.
              "actions": [
                {
                  "setTimer": {
                    "timerName": "coolingTimer",
```

```
                    "seconds": 180
                  }
                },
                {
                  "setVariable": {
                    // We use this 'goodToGo' variable to store the status of the timer
expiration
                    //   for use in conditions that also use input variable values. If
                    //   'timeout()' is used in such mixed conditionals, its value is
lost.
                    "variableName": "goodToGo",
                    "value": "false"
                  }
                }
              ]
            },
            {
              "eventName": "dontDelay",
              "condition": "$variable.noDelay == true",
              // If the heating/cooling unit shutoff delay is not used, no need to
wait.
              "actions": [
                {
                  "setVariable": {
                    "variableName": "goodToGo",
                    "value": "true"
                  }
                }
              ]
            },
            {
              "eventName": "beenHere",
              "condition": "true",
              "actions": [
                {
                  "setVariable": {
                    "variableName": "enteringNewState",
                    "value": "false"
                  }
                }
              ]
            }
          ]
        },
```

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

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

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

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

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

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

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


    {
      "stateName": "heating",
      "onEnter": {
        "events": [
          {
```

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

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

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

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

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

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

          {
```

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

    ],

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

응답:

```
{
    "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"
    }
}
```

BatchUpdateDetector를 사용하여 AWS IoT Events 감지기 모델 업데이트

`BatchUpdateDetector` 작업을 사용하여 타이머 및 변수 값을 포함하여 감지기 인스턴스를 알려진 상태로 만들 수 있습니다. 다음 예제에서 `BatchUpdateDetector` 작업은 온도 모니터링 및 제어 중인 영역의 작동 파라미터를 재설정합니다. 이 작업을 통해 감지기 모델을 삭제, 재생성 또는 업데이트할 필요 없이 이를 수행할 수 있습니다.

CLI 명령:

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

파일: `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": [
```

```
        ]
      }
    }
  ]
}
```

응답:

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

의 입력에 BatchPutMessage 사용 AWS IoT Events

Example 1

BatchPutMessage 작업을 사용하여 온도 제어 및 모니터링 중인 특정 영역에 대한 작동 파라미터를 설정하는 "seedTemperatureInput" 메시지를 보낼 수 있습니다. 새가 AWS IoT Events 있는에서 수신한 모든 메시지로 "areaId" 인해 새 감지기 인스턴스가 생성됩니다. 하지만 새 감지기 인스턴스는 새로운 구역에 대한 "seedTemperatureInput" 메시지가 수신될 때까지 "idle"로 상태를 변경하거나 온도 모니터링과 난방 또는 냉방 장치 제어를 시작하지 않습니다.

CLI 명령:

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

파일: 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}"
    }
  ]
}
```

응답:

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

Example

2

BatchPutMessage 작업을 통해 "temperatureInput" 메시지를 전송하여 지정된 제어 및 모니터링 영역에 있는 센서의 온도 센서 데이터를 보고할 수 있습니다.

CLI 명령:

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

파일: temperatureExample.json

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

응답:

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

Example 3

BatchPutMessage 작업을 통해 "seedTemperatureInput" 메시지를 전송하여 주어진 구역에 대해 원하는 온도 값을 변경할 수 있습니다.

CLI 명령:

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

파일: seedSetDesiredTemp.json

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

응답:

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

에서 MQTT 메시지 수집 AWS IoT Events

센서 컴퓨팅 리소스가 "BatchPutMessage" API를 사용할 수 없지만 경량 MQTT 클라이언트를 사용하여 AWS IoT Core 메시지 브로커로 데이터를 전송할 수 있는 경우 AWS IoT Core 주제 규칙을 생성하여 메시지 데이터를 AWS IoT Events 입력으로 리디렉션할 수 있습니다. 다음은 MQTT AWS IoT Events 주제의 "areaId" 및 "sensorId" 입력 필드와 메시지 페이로드 "sensorData.temperature" 필드의 "temp" 필드를 가져와이 데이터를 로 수집하는 주제 규칙의 정의입니다 AWS IoT Events "temperatureInput".

CLI 명령:

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

파일: seedSetDesiredTemp.json

```
{
```

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

응답: [없음]

센서가 다음 페이로드와 함께 주제 "update/temperature/Area51/03"에 대한 메시지를 보내는 경우.

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

이렇게 하면 다음 "BatchPutMessage" API 호출이 이루어진 AWS IoT Events 것처럼 데이터가에 수집됩니다.

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

파일: spoofExample.json

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

```
  ]
}
```

에서 Amazon SNS 메시지 생성 AWS IoT Events

다음은 "Area51" 감지기 인스턴스에서 생성된 SNS 메시지의 예입니다.

AWS IoT Events 는 Amazon SNS와 통합하여 감지된 이벤트를 기반으로 알림을 생성하고 게시할 수 있습니다. 이 섹션에서는 AWS IoT Events 감지기 인스턴스, 특히 "Area51" 감지기가 Amazon SNS 메시지를 생성하는 방법을 보여줍니다. 이 예제에서는 AWS IoT Events 감지기 내의 다양한 상태 및 이벤트에 의해 트리거되는 Amazon SNS 알림의 구조와 콘텐츠를 보여 주며, 실시간 알림 및 통신을 위해 Amazon SNS AWS IoT Events 와를 결합하는 기능을 보여줍니다.

```
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"}
```

에서 DescribeDetector API 구성 AWS IoT Events

의 DescribeDetector API AWS IoT Events 를 사용하면 특정 감지기 인스턴스에 대한 세부 정보를 검색할 수 있습니다. 이 작업은 감지기의 현재 상태, 변수 값 및 활성 타이머에 대한 인사이트를 제공합니다. 이 API를 사용하면 AWS IoT Events 탐지기의 실시간 상태를 모니터링하여 IoT 이벤트 처리 워크플로의 디버깅, 분석 및 관리를 용이하게 할 수 있습니다.

CLI 명령:

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

응답:


```
{
    "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"
    }
}
```

에 규칙 AWS IoT Core 엔진 사용 AWS IoT Events

다음 규칙은 AWS IoT Core MQTT 메시지를 섀도우 업데이트 요청 메시지로 다시 게시합니다. 감지기 모델에 의해 제어되는 각 영역의 난방 장치 및 냉각 장치에 AWS IoT Core 사물이 정의되어 있다고 가정합니다. 이 예제에서는 이름이 "Area51HeatingUnit" 및 "Area51CoolingUnit"인 항목을 정의했습니다.

CLI 명령:

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

파일: ADMShadowCoolOffRule.json

```
{
  "ruleName": "ADMShadowCoolOff",
  "topicRulePayload": {
    "sql": "SELECT topic(3) as state.desired.command FROM 'hvac/Cooling/Off'",
    "description": "areaDetectorModel mqtt topic publish to cooling unit shadow
 request",
    "ruleDisabled": false,
    "awsIotSqlVersion": "2016-03-23",
    "actions": [
      {
        "republish": {
          "topic": "$$aws/things/${payload.detector.keyValue}CoolingUnit/shadow/
update",
          "roleArn": "arn:aws:iam::123456789012:role/service-role/ADMShadowRole"
        }
      }
    ]
  }
}
```

응답: [비어 있음]

CLI 명령:

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

파일: ADMShadowCoolOnRule.json

```
{
  "ruleName": "ADMShadowCoolOn",
  "topicRulePayload": {
    "sql": "SELECT topic(3) as state.desired.command FROM 'hvac/Cooling/On'",
    "description": "areaDetectorModel mqtt topic publish to cooling unit shadow
 request",
    "ruleDisabled": false,
    "awsIotSqlVersion": "2016-03-23",
    "actions": [
      {
        "republish": {
          "topic": "$$aws/things/${payload.detector.keyValue}CoolingUnit/shadow/
update",
          "roleArn": "arn:aws:iam::123456789012:role/service-role/ADMShadowRole"
        }
      }
    ]
  }
}
```

응답: [비어 있음]

CLI 명령:

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

파일: 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"
        }
      }
    ]
  }
}
```

응답: [비어 있음]

CLI 명령:

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

파일: 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"
        }
      }
    ]
  }
}
```

응답: [비어 있음]

에서 데이터를 수신하고 작업을 트리거하는 데 지원되는 작업 AWS IoT Events

AWS IoT Events 는 지정된 이벤트 또는 전환 이벤트를 감지하면 작업을 트리거할 수 있습니다. 기본 작업을 정의하여 타이머를 사용하거나 변수를 설정하거나 데이터를 다른 AWS 리소스로 보낼 수 있습니다. 이러한 작업을 구성하고 사용자 지정하여 다양한 IoT 이벤트에 대한 자동 응답을 생성하는 방법을 알아봅니다.

> **Note**
>
> 감지기 모델에서 동작을 정의할 때 문자열 데이터 유형인 파라미터에 대한 표현식을 사용할 수 있습니다. 자세한 내용은 표현식을 참조하십시오.

AWS IoT Events 는 타이머를 사용하거나 변수를 설정할 수 있는 다음 작업을 지원합니다.

- `setTimer`로 타이머 생성.
- `resetTimer`로 타이머 재설정.
- `clearTimer`로 타이머 삭제.
- `setVariable`로 변수 생성.

AWS IoT Events 는 AWS 서비스를 사용할 수 있는 다음 작업을 지원합니다.

- `iotTopicPublish`로 MQTT 주제에 대한 메시지 게시.
- `iotEvents`로 AWS IoT Events 에 입력 값으로 데이터 전송.
- `iotSiteWise`로 AWS IoT SiteWise의 자산 속성에 데이터를 보냅니다.
- `dynamoDB`로 Amazon DynamoDB 테이블에 데이터 전송.
- `dynamoDBv2`로 Amazon DynamoDB 테이블에 데이터 전송.
- `firehose` Amazon Data Firehose 스트림으로 데이터를 전송합니다.
- `lambda`로 AWS Lambda 함수 호출.
- `sns`로 푸시 알림에 데이터 전송.
- `sqs`로 Amazon SQS 대기열에 데이터 전송.

AWS IoT Events 기본 제공 타이머 및 변수 작업 사용

AWS IoT Events 는 타이머를 사용하거나 변수를 설정할 수 있는 다음 작업을 지원합니다.

- `setTimer`로 타이머 생성.
- `resetTimer`로 타이머 재설정.
- `clearTimer`로 타이머 삭제.
- `setVariable`로 변수 생성.

타이머 작업 설정

Set timer action

`setTimer` 작업을 사용하면 초 단위로 지속되는 타이머를 만들 수 있습니다.

More information (2)

타이머를 생성할 때 다음 파라미터를 지정해야 합니다.

`timerName`

타이머의 이름입니다.

`durationExpression`

(선택 사항) 타이머의 지속 시간(초)입니다.

지속 시간 표현식의 평가된 결과는 가장 가까운 정수로 내림됩니다. 예를 들어 타이머를 60.99초로 설정하면 기간 표현식의 평가 결과는 60초입니다.

자세한 내용을 알아보려면 AWS IoT Events API 참조의 SetTimerAction를 참조하십시오.

타이머 작업 재설정

Reset timer action

`resetTimer` 작업을 사용하면 타이머를 이전에 평가된 지속 시간 표현식 결과로 설정할 수 있습니다.

More information (1)

타이머를 재설정할 때 다음 파라미터를 지정해야 합니다.

`timerName`

타이머의 이름입니다.

AWS IoT Events 는 타이머를 재설정할 때 기간 표현식을 재평가하지 않습니다.

자세한 내용은 AWS IoT Events API 참조의 ResetTimerAction을 참조하십시오.

타이머 작업 지우기

Clear timer action

`clearTimer` 작업을 사용하면 기존 타이머를 삭제할 수 있습니다.

More information (1)

타이머를 삭제할 때 다음 파라미터를 지정해야 합니다.

`timerName`

타이머의 이름입니다.

자세한 내용은 AWS IoT Events API 참조의 ClearTimerAction을 참조하십시오.

변수 작업 설정

Set variable action

`setVariable` 작업을 사용하면 지정된 값으로 변수를 만들 수 있습니다.

More information (2)

변수를 생성할 때 다음 파라미터를 지정해야 합니다.

`variableName`

변수의 이름입니다.

`value`

변수의 새 값입니다.

자세한 내용을 알아보려면 AWS IoT Events API 참조의 SetVariableAction을 참조하십시오.

AWS IoT Events 다른 AWS 서비스 작업

AWS IoT Events 는 AWS 서비스를 사용할 수 있는 다음 작업을 지원합니다.

- `iotTopicPublish`로 MQTT 주제에 대한 메시지 게시.
- `iotEvents`로 AWS IoT Events 에 입력 값으로 데이터 전송.
- `iotSiteWise`로 AWS IoT SiteWise의 자산 속성에 데이터를 보냅니다.
- `dynamoDB`로 Amazon DynamoDB 테이블에 데이터 전송.
- `dynamoDBv2`로 Amazon DynamoDB 테이블에 데이터 전송.
- `firehose` Amazon Data Firehose 스트림으로 데이터를 전송합니다.
- `lambda`로 AWS Lambda 함수 호출.
- `sns`로 푸시 알림에 데이터 전송.
- `sqs`로 Amazon SQS 대기열에 데이터 전송.

> **Important**
>
> - AWS IoT Events 및 AWS 서비스에서 모두 사용할 동일한 AWS 리전을 선택해야 합니다. 지원되는 리전 목록은 Amazon Web Services 일반 참조에서 AWS IoT Events 엔드포인트 및 할당량을 참조하세요.
> - AWS IoT Events 작업에 대한 다른 AWS 리소스를 생성할 때 동일한 AWS 리전을 사용해야 합니다. AWS 리전을 전환하면 AWS 리소스에 액세스하는 데 문제가 있을 수 있습니다.

기본적으로는 모든 작업에 대해 표준 페이로드를 JSON으로 AWS IoT Events 생성합니다. 이 작업 페이로드에는 작업을 트리거한 이벤트와 감지기 모델 인스턴스에 대한 정보가 있는 모든 속성-값 페어가 포함됩니다. 작업 페이로드를 구성하려면 콘텐츠 표현식을 사용하면 됩니다. 자세한 내용은 AWS IoT Events API 참조의 이벤트 데이터를 필터링, 변환 및 처리하는 표현식 및 Payload를 참조하십시오.

AWS IoT Core

IoT topic publish action

AWS IoT Core 작업을 통해 메시지 브로커를 통해 MQTT AWS IoT 메시지를 게시할 수 있습니다. 지원되는 리전 목록은 Amazon Web Services 일반 참조에서 AWS IoT Core 엔드포인트 및 할당량을 참조하세요.

AWS IoT 메시지 브로커는 게시 AWS IoT 클라이언트에서 구독 클라이언트로 메시지를 전송하여 클라이언트를 연결합니다. 자세한 내용은 AWS IoT 개발자 안내서의 디바이스 통신 프로토콜을 참조하세요.

More information (2)

MQTT 메시지를 게시할 때 다음 파라미터를 지정해야 합니다.

`mqttTopic`

메시지를 수신하는 MQTT 주제입니다.

감지기 모델에서 생성된 변수 또는 입력 값을 사용하여 런타임에 MQTT 주제 이름을 동적으로 정의할 수 있습니다.

`payload`

(선택 사항) 기본 페이로드에는 작업을 트리거한 이벤트와 감지기 모델 인스턴스에 대한 정보가 있는 모든 속성-값 페어가 포함됩니다. 또한 페이로드를 사용자 지정할 수도 있습니다. 자세한 내용은 AWS IoT Events API 참조의 Payload를 참조하십시오.

> **Note**
>
> AWS IoT Events 서비스 역할에 연결된 정책이 `iot:Publish` 권한을 부여하는지 확인합니다. 자세한 내용은 에 대한 자격 증명 및 액세스 관리 AWS IoT Events 단원을 참조하십시오.

자세한 내용은 AWS IoT Events API 참조의 IotTopicPublishAction을 참조하십시오.

AWS IoT Events

IoT Events action

AWS IoT Events 작업을 통해에 데이터를 입력 AWS IoT Events 으로 전송할 수 있습니다. 지원되는 리전 목록은 Amazon Web Services 일반 참조에서 AWS IoT Events 엔드포인트 및 할당량을 참조하세요.

AWS IoT Events 를 사용하면 장비 또는 디바이스 플릿의 장애 또는 작동 변경을 모니터링하고 이러한 이벤트가 발생할 때 작업을 트리거할 수 있습니다. 자세한 내용은 AWS IoT Events 개발자 안내서의 What is AWS IoT Events?를 참조하세요.

More information (2)

데이터를 전송할 때 다음 파라미터를 지정 AWS IoT Events해야 합니다.

`inputName`

데이터를 수신하는 AWS IoT Events 입력의 이름입니다.

`payload`

(선택 사항) 기본 페이로드에는 작업을 트리거한 이벤트와 감지기 모델 인스턴스에 대한 정보가 있는 모든 속성-값 페어가 포함됩니다. 또한 페이로드를 사용자 지정할 수도 있습니다. 자세한 내용은 AWS IoT Events API 참조의 Payload를 참조하십시오.

> **Note**
>
> AWS IoT Events 서비스 역할에 연결된 정책이 `iotevents:BatchPutMessage` 권한을 부여하는지 확인합니다. 자세한 내용은 에 대한 자격 증명 및 액세스 관리 AWS IoT Events 단원을 참조하십시오.

자세한 내용은 AWS IoT Events API 참조의 IotEventsAction을 참조하십시오.

AWS IoT SiteWise

IoT SiteWise action

AWS IoT SiteWise 작업을 통해의 자산 속성으로 데이터를 전송할 수 있습니다 AWS IoT SiteWise. 지원되는 리전 목록은 Amazon Web Services 일반 참조에서 AWS IoT SiteWise 엔드포인트 및 할당량을 참조하세요.

AWS IoT SiteWise 는 대규모 산업 장비에서 데이터를 수집, 구성 및 분석할 수 있는 관리형 서비스입니다. 자세한 내용은 AWS IoT SiteWise사용 설명서의 AWS IoT SiteWise 이란?을 참조하세요.

More information (11)

의 자산 속성으로 데이터를 전송할 때 다음 파라미터를 지정 AWS IoT SiteWise해야 합니다.

> **Important**
>
> 데이터를 수신하려면 AWS IoT SiteWise에서 기존 자산 속성을 사용해야 합니다.
>
> - AWS IoT Events 콘솔을 사용하는 경우를 지정`propertyAlias`하여 대상 자산 속성을 식별해야 합니다.
> - 를 사용하는 경우 대상 자산 속성을 식별`propertyId`하려면 `assetId` 및 중 하나 `propertyAlias` 또는 둘 다를 지정해야 AWS CLI합니다.
>
> 자세한 내용을 알아보려면 AWS IoT SiteWise 사용 설명서의 산업 데이터 스트림을 자산 속성에 매핑를 참조하십시오.

`propertyAlias`

(선택 사항) 자산 속성 목록입니다. 표현식을 지정할 수도 있습니다.

`assetId`

(선택 사항) 지정된 속성이 있는 자산의 ID입니다. 표현식을 지정할 수도 있습니다.

`propertyId`

(선택 사항) 자산 속성 ID입니다. 표현식을 지정할 수도 있습니다.

entryId

(선택 사항) 이 항목의 고유 식별자입니다. 항목 ID를 사용하여 실패 시 오류를 발생시키는 데이터 항목을 추적할 수 있습니다. 기본값은 새 고유 식별자입니다. 표현식을 지정할 수도 있습니다.

propertyValue

속성값에 대한 세부 정보를 포함하는 구조입니다.

quality

(선택 사항) 자산 속성 값의 품질입니다. 값은 GOOD, BAD 또는 UNCERTAIN이어야 합니다. 표현식을 지정할 수도 있습니다.

timestamp

(선택 사항) 타임스탬프 정보가 포함된 구조입니다. 이 값을 지정하지 않으면 기본값은 이벤트 시간입니다.

timeInSeconds

Unix epoch 형식의 타임스탬프(초)입니다. 유효한 범위는 1~31556889864403199입니다. 표현식을 지정할 수도 있습니다.

offsetInNanos

(선택 사항) timeInSeconds에서 변환된 나노초의 오프셋입니다. 유효한 범위는 0~999999999입니다. 표현식을 지정할 수도 있습니다.

value

자산 속성 값이 포함된 구조입니다.

> **Important**
>
> 지정된 자산 속성의 dataType에 따라 다음 값 유형 중 하나를 지정해야 합니다. 자세한 내용은 AWS IoT SiteWise API 참조의 AssetProperty을 참조하십시오.

booleanValue

(선택 사항) 자산 속성 값은 부울 값이며 TRUE 또는 FALSE여야 합니다. 표현식을 지정할 수도 있습니다. 표현식을 사용하는 경우 평가된 결과는 부울 값이어야 합니다.

`doubleValue`

(선택 사항) 자산 속성 값은 실수입니다. 표현식을 지정할 수도 있습니다. 표현식을 사용하는 경우 평가된 결과는 실수여야 합니다.

`integerValue`

(선택 사항) 자산 속성 값은 정수입니다. 표현식을 지정할 수도 있습니다. 표현식을 사용하는 경우 평가된 결과는 정수여야 합니다.

`stringValue`

(선택 사항) 자산 속성 값은 문자열입니다. 표현식을 지정할 수도 있습니다. 표현식을 사용하는 경우 평가된 결과는 문자열이어야 합니다.

> **Note**
>
> AWS IoT Events 서비스 역할에 연결된 정책이 `iotsitewise:BatchPutAssetPropertyValue` 권한을 부여하는지 확인합니다. 자세한 내용은 에 대한 자격 증명 및 액세스 관리 AWS IoT Events 단원을 참조하십시오.

자세한 내용은 AWS IoT Events API 참조의 IotSiteWiseAction을 참조하십시오.

Amazon DynamoDB

DynamoDB action

Amazon DynamoDB 작업을 통해 데이터를 DynamoDB 테이블로 보낼 수 있습니다. DynamoDB 테이블의 한 열에는 지정한 작업 페이로드의 모든 속성-값 페어가 수신됩니다. 지원하는 리전 목록은 Amazon Web Services 일반 참조의 Amazon DynamoDB 엔드포인트 및 할당량을 참조하십시오.

Amazon DynamoDB는 완전관리형 NoSQL 데이터베이스 서비스로서 원활한 확장성과 함께 빠르고 예측 가능한 성능을 제공합니다. 자세한 내용을 알아보려면 Amazon DynamoDB 개발자 안내서의 What Is DynamoDB?를 참조하십시오.

More information (10)

DynamoDB 테이블의 한 열로 데이터를 보낼 때는 다음 파라미터를 지정해야 합니다.

tableName

데이터를 수신하는 DynamoDB 테이블의 이름입니다. tableName 값은 DynamoDB 테이블의 테이블 이름과 일치해야 합니다. 표현식을 지정할 수도 있습니다.

hashKeyField

해시 키(파티션 키라고도 함)의 이름입니다. hashKeyField 값은 DynamoDB 테이블의 파티션 키와 일치해야 합니다. 표현식을 지정할 수도 있습니다.

hashKeyType

(선택 사항) 해시 키의 데이터 형식입니다. 해시 키 유형의 값은 STRING 또는 NUMBER이어야 합니다. 기본값은 STRING입니다. 표현식을 지정할 수도 있습니다.

hashKeyValue

해시 키의 값입니다. hashKeyValue은(는) 대체 템플릿을 사용합니다. 이러한 템플릿은 런타임 시 데이터를 제공합니다. 표현식을 지정할 수도 있습니다.

rangeKeyField

(선택 사항) 범위 키(정렬 키라고도 함)의 이름입니다. rangeKeyField 값은 DynamoDB 테이블의 정렬 키와 일치해야 합니다. 표현식을 지정할 수도 있습니다.

rangeKeyType

(선택 사항) 범위 키의 데이터 유형입니다. 해시 키 유형의 값은 STRING 또는 NUMBER이어야 합니다. 기본값은 STRING입니다. 표현식을 지정할 수도 있습니다.

rangeKeyValue

(선택 사항) 범위 키의 값입니다. rangeKeyValue은(는) 대체 템플릿을 사용합니다. 이러한 템플릿은 런타임 시 데이터를 제공합니다. 표현식을 지정할 수도 있습니다.

작업

(선택 사항) 수행할 작업의 유형입니다. 표현식을 지정할 수도 있습니다. 작업 값은 다음 중 하나여야 합니다.

- INSERT - 데이터를 새 항목으로 DynamoDB 테이블에 삽입합니다. 이것이 기본값입니다.
- UPDATE - DynamoDB 테이블의 기존 항목을 새 데이터로 업데이트합니다.
- DELETE - DynamoDB 테이블의 기존 항목을 삭제합니다.

`payloadField`

(선택 사항) 작업 페이로드를 수신하는 DynamoDB 열의 이름입니다. 기본 이름은 `payload`입니다. 표현식을 지정할 수도 있습니다.

`payload`

(선택 사항) 기본 페이로드에는 작업을 트리거한 이벤트와 감지기 모델 인스턴스에 대한 정보가 있는 모든 속성-값 페어가 포함됩니다. 또한 페이로드를 사용자 지정할 수도 있습니다. 자세한 내용은 AWS IoT Events API 참조의 Payload를 참조하십시오.

특정 페이로드 유형이 문자열인 경우 `DynamoDBAction`은 비 JSON 데이터를 이진 데이터로 DynamoDB 테이블에 보냅니다. DynamoDB 콘솔은 데이터를 Base64 인코딩된 텍스트로 표시합니다. `payloadField` 값은 *`payload-field`*`_raw`입니다. 표현식을 지정할 수도 있습니다.

> **Note**
>
> AWS IoT Events 서비스 역할에 연결된 정책이 `dynamodb:PutItem` 권한을 부여하는지 확인합니다. 자세한 내용은 에 대한 자격 증명 및 액세스 관리 AWS IoT Events 단원을 참조하십시오.

자세한 내용은 AWS IoT Events API 참조의 DynamoDBAction을 참조하십시오.

Amazon DynamoDB(v2)

DynamoDBv2 action

Amazon DynamoDB(v2) 작업을 사용하면 DynamoDB 테이블에 데이터를 쓸 수 있습니다. DynamoDB 테이블의 별도 열에는 지정한 작업 페이로드의 속성-값 페어 하나가 수신됩니다. 지원하는 리전 목록은 Amazon Web Services 일반 참조의 Amazon DynamoDB 엔드포인트 및 할당량을 참조하십시오.

Amazon DynamoDB는 완전관리형 NoSQL 데이터베이스 서비스로서 원활한 확장성과 함께 빠르고 예측 가능한 성능을 제공합니다. 자세한 내용을 알아보려면 Amazon DynamoDB 개발자 안내서의 What Is DynamoDB?를 참조하십시오.

More information (2)

DynamoDB 테이블의 여러 열로 데이터를 보내는 경우 다음 파라미터를 지정해야 합니다.

`tableName`

데이터를 수신하는 DynamoDB 테이블의 이름입니다. 표현식을 지정할 수도 있습니다.

`payload`

(선택 사항) 기본 페이로드에는 작업을 트리거한 이벤트와 감지기 모델 인스턴스에 대한 정보가 있는 모든 속성-값 페어가 포함됩니다. 또한 페이로드를 사용자 지정할 수도 있습니다. 자세한 내용은 AWS IoT Events API 참조의 Payload를 참조하십시오.

> **Important**
>
> 페이로드 유형은 JSON이어야 합니다. 표현식을 지정할 수도 있습니다.

> **Note**
>
> AWS IoT Events 서비스 역할에 연결된 정책이 `dynamodb:PutItem` 권한을 부여하는지 확인합니다. 자세한 내용은 에 대한 자격 증명 및 액세스 관리 AWS IoT Events 단원을 참조하십시오.

자세한 내용은 AWS IoT Events API 참조의 DynamoDBv2Action을 참조하십시오.

Amazon Data Firehose

Firehose action

Amazon Data Firehose 작업을 사용하면 Firehose 전송 스트림으로 데이터를 전송할 수 있습니다. 지원되는 리전 목록은의 Amazon Data Firehose 엔드포인트 및 할당량을 참조하세요Amazon Web Services 일반 참조.

Amazon Data Firehose는 Amazon Simple Storage Service(Amazon Simple Storage Service), Amazon Redshift, Amazon OpenSearch Service(OpenSearch Service) 및 Splunk와 같은 대상으로 실시간 스트리밍 데이터를 제공하기 위한 완전관리형 서비스입니다. 자세한 내용은 Amazon Data Firehose 개발자 안내서의 Amazon Data Firehose란?을 참조하세요.

More information (3)

Firehose 전송 스트림으로 데이터를 전송할 때는 다음 파라미터를 지정해야 합니다.

deliveryStreamName

데이터를 수신하는 Firehose 전송 스트림의 이름입니다.

separator

(선택 사항) 문자 구분자를 사용하여 Firehose 전송 스트림으로 전송되는 연속 데이터를 분리할 수 있습니다. 구분자 값은 '\n'(줄 바꿈), '\t'(탭), '\r\n'(Windows 새 줄) 또는 ','(쉼표)여야 합니다.

payload

(선택 사항) 기본 페이로드에는 작업을 트리거한 이벤트와 감지기 모델 인스턴스에 대한 정보가 있는 모든 속성-값 페어가 포함됩니다. 또한 페이로드를 사용자 지정할 수도 있습니다. 자세한 내용은 AWS IoT Events API 참조의 Payload를 참조하십시오.

> **Note**
>
> AWS IoT Events 서비스 역할에 연결된 정책이 `firehose:PutRecord` 권한을 부여하는지 확인합니다. 자세한 내용은 에 대한 자격 증명 및 액세스 관리 AWS IoT Events 단원을 참조하십시오.

자세한 내용은 AWS IoT Events API 참조의 FirehoseAction을 참조하십시오.

AWS Lambda

Lambda action

AWS Lambda 작업을 통해 Lambda 함수를 호출할 수 있습니다. 지원되는 리전 목록은 Amazon Web Services 일반 참조에서 AWS Lambda 엔드포인트 및 할당량을 참조하세요.

AWS Lambda 는 서버를 프로비저닝하거나 관리하지 않고도 코드를 실행할 수 있는 컴퓨팅 서비스입니다. 자세한 내용은 AWS Lambda 개발자 안내서의 What is AWS Lambda?를 참조하세요.

More information (2)

Lambda 함수를 호출할 때 다음 파라미터를 지정해야 합니다.

functionArn

호출할 Lambda 함수의 ARN입니다.

`payload`

(선택 사항) 기본 페이로드에는 작업을 트리거한 이벤트와 감지기 모델 인스턴스에 대한 정보가 있는 모든 속성-값 페어가 포함됩니다. 또한 페이로드를 사용자 지정할 수도 있습니다. 자세한 내용은 AWS IoT Events API 참조의 Payload를 참조하십시오.

> **Note**
>
> AWS IoT Events 서비스 역할에 연결된 정책이 `lambda:InvokeFunction` 권한을 부여하는지 확인합니다. 자세한 내용은 에 대한 자격 증명 및 액세스 관리 AWS IoT Events 단원을 참조하십시오.

자세한 내용은 AWS IoT Events API 참조의 LambdaAction을 참조하십시오.

Amazon Simple Notification Service

SNS action

Amazon SNS 주제 게시 작업을 사용하면 Amazon SNS 메시지를 게시할 수 있습니다. 지원하는 리전의 목록은 Amazon Web Services 일반 참조의 Amazon Simple Notification Service 엔드포인트 및 할당량을 참조하십시오.

Amazon Simple Notification Service(Amazon SNS)는 구독 중인 엔드포인트 또는 클라이언트에 대한 메시지 전달 또는 전송을 조정 및 관리하는 웹 서비스입니다. 자세한 정보는 Amazon Simple Notification 개발자 안내서의 What is Amazon SNS?를 참조하십시오.

> **Note**
>
> Amazon SNS 주제 게시 작업은 Amazon SNS FIFO(선입선출)를 지원하지 않습니다. 규칙 엔진은 완전 분산형 서비스이므로 Amazon SNS 작업이 시작될 때 메시지가 지정된 순서로 표시되지 않을 수 있습니다.

More information (2)

Amazon SNS 메시지를 게시할 때 다음 파라미터를 지정해야 합니다.

`targetArn`

메시지가 전송되는 Amazon SNS 대상의 ARN입니다.

`payload`

(선택 사항) 기본 페이로드에는 작업을 트리거한 이벤트와 감지기 모델 인스턴스에 대한 정보가 있는 모든 속성-값 페어가 포함됩니다. 또한 페이로드를 사용자 지정할 수도 있습니다. 자세한 내용은 AWS IoT Events API 참조의 Payload를 참조하십시오.

> **Note**
>
> AWS IoT Events 서비스 역할에 연결된 정책이 `sns:Publish` 권한을 부여하는지 확인합니다. 자세한 내용은 에 대한 자격 증명 및 액세스 관리 AWS IoT Events 단원을 참조하십시오.

자세한 내용은 AWS IoT Events API 참조의 SNSTopicPublishAction을 참조하십시오.

Amazon Simple Queue Service

SQS action

Amazon SQS 작업을 통해 Amazon SQS 대기열로 전송할 수 있습니다. 지원되는 리전 목록은 Amazon Web Services 일반 참조의 Amazon Simple Queue Service 엔드포인트 및 할당량을 참조하십시오.

Amazon Simple Queue Service(Amazon SQS)는 내구력 있고 가용성이 뛰어난 보안 호스팅 대기열을 제공하며 이를 통해 분산 소프트웨어 시스템과 구성 요소를 통합 및 분리할 수 있습니다. 자세한 정보는 Amazon Simple Queue Service 개발자 안내서의 What is Amazon Simple Queue Service?를 참조하십시오.

> **Note**
>
> Amazon SQS 작업은 >Amazon SQS FIFO(선입선출) 주제를 지원하지 않습니다. 규칙 엔진은 완전 분산형 서비스이므로 Amazon SQS 작업이 시작될 때 메시지가 지정된 순서로 표시되지 않을 수 있습니다.

More information (3)

Amazon SQS 대기열로 전송할 때 다음 파라미터를 지정해야 합니다.

`queueUrl`

데이터를 수신하는 Amazon SQS 대기열의 URL입니다.

`useBase64`

(선택 사항)을 지정하는 경우 데이터를 Base64 텍스트로 AWS IoT Events 인코딩합니다`TRUE`. 기본값은 `FALSE`입니다.

`payload`

(선택 사항) 기본 페이로드에는 작업을 트리거한 이벤트와 감지기 모델 인스턴스에 대한 정보가 있는 모든 속성-값 페어가 포함됩니다. 또한 페이로드를 사용자 지정할 수도 있습니다. 자세한 내용은 AWS IoT Events API 참조의 Payload를 참조하십시오.

> **Note**
>
> AWS IoT Events 서비스 역할에 연결된 정책이 `sqs:SendMessage` 권한을 부여하는지 확인합니다. 자세한 내용은 에 대한 자격 증명 및 액세스 관리 AWS IoT Events 단원을 참조하십시오.

자세한 내용은 AWS IoT Events API 참조의 SNSTopicPublishAction을 참조하십시오.

Amazon SNS와 AWS IoT Core 규칙 엔진을 사용하여 함수를 트리거할 수도 있습니다 AWS Lambda . 이를 통해 Amazon Connect와 같은 다른 서비스나 회사 ERP(전사적 자원 관리) 애플리케이션을 사용하여 조치를 취할 수 있습니다.

> **Note**
>
> Amazon Amazon Kinesis와 같은 다른 AWS 서비스를 사용하여 대규모 데이터 레코드 스트림을 실시간으로 수집하고 처리할 수 있습니다. 여기에서 초기 분석을 완료한 다음 결과를에 감지기에 대한 입력 AWS IoT Events 으로 보낼 수 있습니다.

이벤트 데이터를 필터링, 변환 및 처리하는 표현식

표현식은 수신 데이터를 평가하고, 계산을 수행하고, 특정 작업 또는 상태 전환이 발생하는 조건을 결정하는 데 사용됩니다.는 감지기 모델을 생성하고 업데이트할 때 값을 지정하는 몇 가지 방법을 AWS IoT Events 제공합니다. 표현식을 사용하여 리터럴 값을 지정하거나 특정 값을 지정하기 전에 표현식을 평가할 AWS IoT Events 수 있습니다.

주제

- 에서 디바이스 데이터를 필터링하고 작업을 정의하는 구문 AWS IoT Events
- 에 대한 표현식 예제 및 사용량 AWS IoT Events

에서 디바이스 데이터를 필터링하고 작업을 정의하는 구문 AWS IoT Events

표현식은 디바이스 데이터를 필터링하고 작업을 정의하는 구문을 제공합니다. AWS IoT Events 표현식에서 리터럴, 연산자, 함수, 참조 및 대체 템플릿을 사용할 수 있습니다. 이러한 구성 요소를 결합하여 강력하고 유연한 표현식을 생성하여 IoT 데이터를 처리하고, 계산을 수행하고, 문자열을 조작하고, 감지기 모델 내에서 논리적 결정을 내릴 수 있습니다.

리터럴

- Integer
- 10진수
- 문자열
- 부울

연산자

단항

- 아니요(부울): !
- 아니요(비트별): ~
- 마이너스(산술): -

문자열

- 연결: **+**

  두 피연산자는 모두 문자열이어야 합니다. 문자열 리터럴은 작은따옴표(')로 묶여야 합니다.

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

Arithmetic

- 더하기: **+**

  두 피연산자는 모두 숫자여야 합니다.

- 빼기: **-**
- 나눗셈: **/**

  피연산자(약수 또는 배당) 중 하나 이상이 10진수가 아닌 경우 나눗셈 결과는 반올림된 정수 값입니다.

- 곱하기: *****

비트별(정수)

- 또는: **|**

  예제: `13 | 5` -> `13`

- 및: **&**

  예제: `13 & 5` -> `5`

- XOR: **^**

  예제: `13 ^ 5` -> `8`

- NOT: **~**

  예제: `~13` -> `-14`

부울

- 작음: **<**
- 작거나 같음: **<=**
- 같음: **==**
- 같지 않음: **!=**
- 크거나 같음: **>=**

- 큼: **>**
- 및: **&&**
- 또는: **||**

> **Note**
>
> ||의 하위 표현식에 정의되지 않은 데이터가 포함된 경우 해당 하위 표현식은 `false`와 같이 취급됩니다.

괄호

괄호를 사용하여 표현식 내의 용어를 그룹화할 수 있습니다.

AWS IoT Events 표현식에 사용할 함수

AWS IoT Events 는 감지기 모델 표현식의 기능을 개선하기 위한 내장 함수 세트를 제공합니다. 이러한 함수를 사용하면 타이머 관리, 유형 변환, null 확인, 트리거 유형 식별, 입력 확인, 문자열 조작 및 비트 단위 작업을 수행할 수 있습니다. 이러한 함수를 활용하면 응답 처리 AWS IoT Events 로직을 생성하여 IoT 애플리케이션의 전반적인 효율성을 개선할 수 있습니다.

기본 제공 함수

`timeout("timer-name")`

지정된 타이머가 경과했는지 여부를 `true` 평가합니다. "*`timer-name`*"을 사용자가 정의한 타이머 이름으로 바꾸어 따옴표로 표기하십시오. 이벤트 작업에서 타이머를 정의한 다음 타이머를 시작하거나, 재설정하거나, 이전에 정의한 타이머를 지울 수 있습니다. 필드 `detectorModelDefinition.states.onInput|onEnter|onExit.events.actions.setTimer.timerName`을 참조하십시오.

한 상태에서 설정된 타이머를 다른 상태에서 참조할 수 있습니다. 타이머가 참조되는 상태를 입력하기 전에 타이머를 만든 상태를 방문해야 합니다.

예를 들어 감지기 모델에는 `TemperatureChecked` 및 라는 두 가지 상태가 있습니다`RecordUpdated`. `TemperatureChecked` 상태에서 타이머를 생성했습니다. `TemperatureChecked` 상태에서 타이머를 사용하려면 먼저 `RecordUpdated` 상태를 방문해야 합니다.

정확성을 보장하기 위해 타이머를 설정해야 하는 시간은 최소 60초입니다.

> **Note**
>
> 실제 타이머 만료 후 처음 확인할 때만 `timeout()`이 `true`로 반환되고 그 이후에는 `false`로 반환됩니다.

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

지정된 유형으로 변환된 표현식의 값으로 평가합니다. *`##`* 값은 `String`, `Boolean` 또는 `Decimal`이어야 합니다. 다음 키워드 중 하나를 사용하거나 키워드가 포함된 문자열로 평가되는 표현식을 사용하십시오. 다음 변환만 성공하여 유효한 값을 반환합니다.

- 부울 -> 문자열

  문자열 `"true"` 또는 `"false"`를 반환합니다.
- 10진수 -> 문자열
- 문자열 -> 부울
- 문자열 -> 10진수

  지정된 문자열은 유효한 10진수 표현이어야 하며, 그렇지 않으면 `convert()`는 실패합니다.

`convert()`가 유효한 값을 반환하지 않으면 해당 값이 속한 표현식도 유효하지 않습니다. 이 결과는 `false`와 동일하며 표현식이 발생한 이벤트의 일부로 `nextState` 지정된 이벤트와 `actions`으로 전환을 트리거하지 않습니다.

`isNull`(*`expression`*)

표현식이 NULL로 반환되면 `true`로 평가됩니다. 예를 들어, 입력 `MyInput`이 메시지 `{ "a": null }`을 수신하면 다음은 `true`로 평가되지만 `isUndefined($input.MyInput.a)`는 `false`로 평가됩니다.

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

`isUndefined`(*`expression`*)

표현식이 정의되지 않았다면 `true`로 평가됩니다. 예를 들어, 입력 `MyInput`이 메시지 `{ "a": null }`을 수신하면 다음은 `false`로 평가되지만 `isNull($input.MyInput.a)`는 `true`로 평가됩니다.

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

triggerType("*type*")

값은 "Message" 또는 "Timer"가 될 수 있습니다. 다음 예제와 같이 타이머가 만료되어 해당 이벤트가 나타나는 이벤트 조건을 평가 중이라면 true로 평가됩니다.

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

또는 입력 메시지가 수신되었습니다.

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

currentInput("*input*")

지정된 입력 메시지를 받았기 때문에 해당 메시지가 나타나는 이벤트 조건을 평가 중이라면 true로 평가됩니다. 예를 들어, 입력 Command가 메시지 { "value": "Abort" }를 수신하면 다음은 true로 평가됩니다.

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

이 함수를 사용하여 다음 표현식에서처럼 특정 입력이 수신되었고 타이머가 만료되지 않았으므로 조건이 평가되고 있는지 확인할 수 있습니다.

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

문자열 일치 함수

startsWith(*expression1*, *expression2*)

첫 번째 문자열 표현식이 두 번째 문자열 표현식으로 시작한다면 true로 평가합니다. 예를 들어, 입력 MyInput이 메시지 { "status": "offline"}을 수신하면 다음은 true로 평가됩니다.

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

두 표현식 모두 문자열 값으로 평가되어야 합니다. 두 표현식 중 하나라도 문자열 값으로 평가되지 않으면 함수의 결과가 정의되지 않습니다. 변환은 수행되지 않습니다.

endsWith(*expression1*, *expression2*)

첫 번째 문자열 표현식이 두 번째 문자열 표현식으로 끝난다면 true로 평가합니다. 예를 들어, 입력 MyInput이 메시지 { "status": "offline" }을 수신하면 다음은 true로 평가됩니다.

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

두 표현식 모두 문자열 값으로 평가되어야 합니다. 두 표현식 중 하나라도 문자열 값으로 평가되지 않으면 함수의 결과가 정의되지 않습니다. 변환은 수행되지 않습니다.

contains(*expression1*, *expression2*)

첫 번째 문자열 표현식에 두 번째 문자열 표현식이 포함되어 있다면 true로 평가합니다. 예를 들어, 입력 MyInput이 메시지 { "status": "offline" }을 수신하면 다음은 true로 평가됩니다.

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

두 표현식 모두 문자열 값으로 평가되어야 합니다. 두 표현식 중 하나라도 문자열 값으로 평가되지 않으면 함수의 결과가 정의되지 않습니다. 변환은 수행되지 않습니다.

비트별 정수 조작 함수

bitor(*expression1*, *expression2*)

정수 표현식의 비트별 OR을 평가합니다(이항 OR 연산은 정수의 대응하는 비트에 대해 수행됨). 예를 들어, 입력 MyInput이 메시지 { "value1": 13, "value2": 5 }를 수신하면 다음은 13으로 평가됩니다.

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

두 표현식 모두 정수 값으로 평가되어야 합니다. 두 표현식 중 하나가 정수 값으로 평가되지 않으면 함수의 결과가 정의되지 않습니다. 변환은 수행되지 않습니다.

bitand(*expression1*, *expression2*)

정수 표현식의 비트별 AND를 평가합니다(이항 AND 연산은 정수의 대응되는 비트에 대해 수행됨). 예를 들어, 입력 MyInput이 메시지 { "value1": 13, "value2": 5 }를 수신하면 다음은 5로 평가됩니다.

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

두 표현식 모두 정수 값으로 평가되어야 합니다. 두 표현식 중 하나가 정수 값으로 평가되지 않으면 함수의 결과가 정의되지 않습니다. 변환은 수행되지 않습니다.

bitxor(*expression1*, *expression2*)

정수 표현식의 비트별 XOR을 평가합니다(이항 XOR 연산은 정수의 대응하는 비트에 대해 수행됨). 예를 들어, 입력 `MyInput`이 메시지 `{ "value1": 13, "value2": 5 }`를 수신하면 다음은 8로 평가됩니다.

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

두 표현식 모두 정수 값으로 평가되어야 합니다. 두 표현식 중 하나가 정수 값으로 평가되지 않으면 함수의 결과가 정의되지 않습니다. 변환은 수행되지 않습니다.

bitnot(*expression*)

정수 표현식의 비트별 NOT을 평가합니다(이항 NOT 연산은 정수 비트에 대해 수행됨). 예를 들어, 입력 `MyInput`이 메시지 `{ "value": 13 }`을 수신하면 다음은 -14로 평가됩니다.

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

두 표현식 모두 정수 값으로 평가되어야 합니다. 두 표현식 중 하나가 정수 값으로 평가되지 않으면 함수의 결과가 정의되지 않습니다. 변환은 수행되지 않습니다.

AWS IoT Events 표현식의 입력 및 변수에 대한 참조

입력

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

`input-name`은(는) 입력 생성 작업을 사용하여 만든 입력입니다.

예를 들어, `inputDefinition.attributes.jsonPath` 항목을 정의한 데 사용할 이름이 지정된 `TemperatureInput` 입력이 있는 경우 다음 사용 가능한 필드에 값이 나타날 수 있습니다.

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

`temperature` 필드 값을 참조하려면 다음 명령을 사용합니다.

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

값이 배열인 필드의 경우 [*n*]을 사용하여 배열의 멤버를 참조할 수 있습니다. 예를 들어 다음과 같은 값이 있다고 가정합니다.

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

다음 명령으로 값 78.8을 참조할 수 있습니다.

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

변수

$variable.*variable-name*

*variable-name*는 CreateDetectorModel 작업을 사용하여 정의한 변수입니다.

예를 들어 detectorDefinition.states.onInputEvents.actions.setVariable.variableName를 사용하여 정의한 이름이 지정된 TechnicianID 변수가 있는 경우 다음 명령을 사용하여 변수에 가장 최근에 지정된(문자열) 값을 참조할 수 있습니다.

```
$variable.TechnicianID
```

setVariable 작업을 사용해서만 변수 값을 설정할 수 있습니다. 표현식에는 변수 값을 할당할 수 없습니다. 변수는 설정을 해제할 수 없습니다. 예를 들어 값 null을 할당할 수 없습니다.

> **Note**
>
> (정규 표현식) 패턴 [a-zA-Z][a-zA-Z0-9_]*를 따르지 않는 식별자를 사용하는 참조에서는 해당 식별자를 백틱(`)으로 묶어야 합니다. 예를 들어 필드 이름이

`$input.MyInput.`_value`로 지정된 입력에 대한 참조는 이 필드 MyInput를 _value로 지정해야 합니다.

표현식에서 참조를 사용하는 경우 다음을 확인하십시오.

- 하나 이상의 연산자와 함께 참조를 피연산자로 사용하는 경우, 참조하는 모든 데이터 유형이 호환되는지 확인하십시오.

  예를 들어, 다음 표현식에서 정수 2는 및 연산자 ==와 && 모두의 피연산자입니다. 피연산자를 호환하기 위해서는 `$variable.testVariable + 1` 및 `$variable.testVariable`이 정수 또는 10진수를 참조해야 합니다.

  또한 정수 1은(는) + 연산자의 피연산자입니다. 따라서 `$variable.testVariable`은(는) 정수 또는 10진수를 참조해야 합니다.

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

- 참조를 함수에 전달된 인수로 사용하는 경우, 함수가 참조하는 데이터 유형을 지원하는지 확인하십시오.

  예를 들어, 다음 `timeout("time-name")` 함수에는 큰따옴표가 있는 문자열이 인수로 필요합니다. *`### ##`* 값에 대한 참조를 사용하는 경우 큰따옴표가 있는 문자열을 참조해야 합니다.

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

  > **Note**
  >
  > `convert(type, expression)` 함수의 경우 *`##`* 값에 대한 참조를 사용하는 경우 참조의 평가 결과는 `String`, `Decimal` 또는 `Boolean`이어야 합니다.

AWS IoT Events 표현식은 정수, 십진수, 문자열 및 부울 데이터 형식을 지원합니다. 다음 표에는 호환되지 않는 유형의 페어 목록이 나와 있습니다.

호환되지 않는 유형 페어
정수, 문자열

호환되지 않는 유형 페어
정수, 부울
10진수, 문자열
10진수, 부울
문자열, 부울

AWS IoT Events 표현식의 대체 템플릿

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

`${}`는 문자열을 보간된 문자열로 식별합니다. 는 모든 AWS IoT Events 표현식일 `expression` 수 있습니다. 여기에는 연산자, 함수 및 참조가 포함됩니다.

예를 들어, SetVariableAction 작업을 사용하여 변수를 정의했습니다. `variableName`은(는) `SensorID`이고 `value`은(는) `10`입니다. 다음과 같은 대체 템플릿을 만들 수 있습니다.

대체 템플릿	결과 문자열
`'${'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}"`

에 대한 표현식 예제 및 사용량 AWS IoT Events

다음과 같은 방식으로 감지기 모델의 값을 지정할 수 있습니다.

- AWS IoT Events 콘솔에 지원되는 표현식을 입력합니다.
- 표현식을 AWS IoT Events APIs에 파라미터로 전달합니다.

표현식은 리터럴, 연산자, 함수, 참조 및 대체 템플릿을 지원합니다.

> **Important**
>
> 표현식은 정수, 10진수, 문자열 또는 부울 값을 참조해야 합니다.

AWS IoT Events 표현식 작성

AWS IoT Events 표현식을 작성하는 데 도움이 되는 다음 예제를 참조하세요.

리터럴

리터럴 값의 경우 표현식에 작은따옴표가 포함되어야 합니다. 부울 값은 `true` 또는 `false` 둘 중 하나입니다.

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

레퍼런스

참조의 경우 변수 또는 입력 값을 지정해야 합니다.

- 다음 입력은 10진수, `10.01`을 참조합니다.

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

- 다음 변수는 문자열, `Greenhouse Temperature Table`을 참조합니다.

  ```
  $variable.TableName
  ```

대체 템플릿

대체 템플릿의 경우 ${}를 사용해야 하며 템플릿은 작은따옴표로 묶어야 합니다. 대체 템플릿은 리터럴, 연산자, 함수, 참조 및 대체 템플릿의 조합을 포함할 수 있습니다.

- 다음 표현식의 평가 결과는 문자열, `50.018 in Fahrenheit`입니다.

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

- 다음 표현식의 평가 결과는 문자열, `{\"sensor_id\":\"Sensor_1\",\"temperature\":\"50.018\"}`입니다.

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

문자열 연결

문자열 연결의 경우 +를 사용해야 합니다. 문자열 연결은 리터럴, 연산자, 함수, 참조 및 대체 템플릿의 조합도 포함할 수 있습니다.

- 다음 표현식의 평가 결과는 문자열, `Greenhouse Temperature Table 2000-01-01`입니다.

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

AWS IoT Events 감지기 모델 예제

이 페이지에서는 다양한 AWS IoT Events 기능을 구성하는 방법을 보여주는 예제 사용 사례 목록을 제공합니다. 이 예제는 온도 임계값과 같은 기본 감지부터 고급 이상 탐지 및 기계 학습 시나리오에 이르기까지 다양합니다. 각 예제에는 AWS IoT Events 탐지, 작업 및 통합을 설정하는 데 도움이 되는 절차와 코드 조각이 포함되어 있습니다. 이 예제에서는 AWS IoT Events 서비스의 유연성과 다양한 IoT 애플리케이션 및 사용 사례에 맞게 서비스를 사용자 지정하는 방법을 보여줍니다. AWS IoT Events 기능을 탐색하거나 특정 탐지 또는 자동화 워크플로를 구현하는 지침이 필요한 경우이 페이지를 참조하세요.

주제

예:에서 HVAC 온도 제어 사용 AWS IoT Events

백그라운드 스토리

이 예제는 다음과 같은 기능을 갖춘 온도 제어 모델(온도 조절기)을 구현합니다.

- 여러 영역을 모니터링하고 제어할 수 있는 하나의 감지기 모델을 정의하십시오. (각 영역에 대해 감지기 인스턴스가 생성됩니다.)
- 각 감지기 인스턴스는 각 제어 영역에 배치된 여러 센서로부터 온도 데이터를 수신합니다.
- 언제든지 각 영역의 원하는 온도(설정점)를 변경할 수 있습니다.
- 각 영역의 작동 파라미터를 정의하고 언제든지 이러한 파라미터를 변경할 수 있습니다.

- 언제든지 영역에 센서를 추가하거나 영역에서 센서를 삭제할 수 있습니다.
- 난방 및 냉방 장치의 최소 작동 시간을 설정하여 손상을 방지할 수 있습니다.
- 감지기는 비정상적인 센서 판독값을 거부하고 보고합니다.
- 비상 온도 설정점을 정의할 수 있습니다. 센서 중 하나에서 사용자가 정의한 설정값보다 높거나 낮은 온도를 보고하면 난방 또는 냉방 장치가 즉시 작동하고 감지기가 온도 급상승을 보고합니다.

이 예제는 다음과 같은 기능을 보여줍니다.

- 이벤트 감지기 모델을 생성합니다.
- 입력을 생성합니다.
- 감지기 모델로 입력을 수집합니다.
- 트리거 조건을 평가하십시오.
- 조건의 상태 변수를 참조하고 조건에 따라 변수 값을 설정합니다.
- 조건의 타이머를 참조하고 조건에 따라 타이머를 설정하십시오.
- Amazon SNS와 MQTT 메시지의 조치를 취하십시오.

의 HVAC 시스템에 대한 입력 정의 AWS IoT Events

`seedTemperatureInput`는 특정 지역의 감지기 인스턴스를 생성하고 해당 영역의 작동 파라미터를 정의하는 데 사용됩니다.

효과적인 기후 제어를 위해서는에서 HVAC 시스템에 대한 입력을 구성하는 AWS IoT Events 것이 중요합니다. 이 예제에서는 온도, 습도, 점유율 및 에너지 소비 데이터와 같은 파라미터를 캡처하는 입력을 설정하는 방법을 보여줍니다. 입력 속성을 정의하고, 데이터 소스를 구성하고, 탐지기 모델이 최적의 관리 및 효율성을 위해 정확하고 시기 적절한 정보를 받을 수 있도록 사전 처리 규칙을 설정하는 방법을 알아봅니다.

CLI 명령은 다음을 사용합니다.

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

파일: `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" }
    ]
  }
}
```

응답:

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

필요에 따라 각 영역의 각 센서가 `temperatureInput`을 전송해야 합니다.

CLI 명령은 다음을 사용합니다.

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

파일: `temperatureInput.json`

```
{
  "inputName": "temperatureInput",
  "inputDescription": "Temperature sensor unit data.",
  "inputDefinition": {
```

```
    "attributes": [
      { "jsonPath": "sensorId" },
      { "jsonPath": "areaId" },
      { "jsonPath": "sensorData.temperature" }
    ]
  }
}
```

응답:

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

를 사용하여 HVAC 시스템에 대한 감지기 모델 정의 AWS IoT Events

`areaDetectorModel`는 각 감지기 인스턴스의 작동 방식을 정의합니다. 각 `state machine` 인스턴스는 온도 센서 판독값을 수집한 다음 이 판독값에 따라 상태를 변경하고 제어 메시지를 보냅니다.

CLI 명령은 다음을 사용합니다.

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

파일: `areaDetectorModel.json`

```
{
  "detectorModelName": "areaDetectorModel",
  "detectorModelDefinition": {
    "states": [
      {
        "stateName": "start",
        "onEnter": {
          "events": [
            {
```

```
            "eventName": "prepare",
            "condition": "true",
            "actions": [
              {
                "setVariable": {
                  "variableName": "sensorId",
                  "value": "0"
                }
              },
              {
                "setVariable": {
                  "variableName": "reportedTemperature",
                  "value": "0.1"
                }
              },
              {
                "setVariable": {
                  "variableName": "resetMe",
                  "value": "false"
                }
              }
            ]
          }
        ]
      },
      "onInput": {
        "transitionEvents": [
          {
            "eventName": "initialize",
            "condition": "$input.seedTemperatureInput.sensorCount > 0",
            "actions": [
              {
                "setVariable": {
                  "variableName": "rangeHigh",
                  "value": "$input.seedTemperatureInput.rangeHigh"
                }
              },
              {
                "setVariable": {
                  "variableName": "rangeLow",
                  "value": "$input.seedTemperatureInput.rangeLow"
                }
              },
              {
```

```
            "setVariable": {
              "variableName": "desiredTemperature",
              "value": "$input.seedTemperatureInput.desiredTemperature"
            }
          },
          {
            "setVariable": {
              "variableName": "averageTemperature",
              "value": "$input.seedTemperatureInput.desiredTemperature"
            }
          },
          {
            "setVariable": {
              "variableName": "allowedError",
              "value": "$input.seedTemperatureInput.allowedError"
            }
          },
          {
            "setVariable": {
              "variableName": "anomalousHigh",
              "value": "$input.seedTemperatureInput.anomalousHigh"
            }
          },
          {
            "setVariable": {
              "variableName": "anomalousLow",
              "value": "$input.seedTemperatureInput.anomalousLow"
            }
          },
          {
            "setVariable": {
              "variableName": "sensorCount",
              "value": "$input.seedTemperatureInput.sensorCount"
            }
          },
          {
            "setVariable": {
              "variableName": "noDelay",
              "value": "$input.seedTemperatureInput.noDelay == true"
            }
          }
        ],
        "nextState": "idle"
      },
```

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

```
            ]
          }
        ]
      }
    },


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

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

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

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

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

```
            },

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

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

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


  {
    "stateName": "cooling",
    "onEnter": {
      "events": [
        {
          "eventName": "delay",
          "condition": "!$variable.noDelay && $variable.enteringNewState",
          "actions": [
            {
              "setTimer": {
                "timerName": "coolingTimer",
                "seconds": 180
              }
            },
            {
              "setVariable": {
                "variableName": "goodToGo",
                "value": "false"
              }
            }
          ]
        },
        {
          "eventName": "dontDelay",
          "condition": "$variable.noDelay == true",
          "actions": [
            {
              "setVariable": {
                "variableName": "goodToGo",
                "value": "true"
```

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

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

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

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

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

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

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

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


      {
```

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

```
            }
          ]
        },

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

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

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

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

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

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

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

    ],

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

응답:

```
{
    "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"
    }
}
```

의 HVAC 시스템에 대한 BatchPutMessage 예제 AWS IoT Events

이 예제에서 BatchPutMessage은(는) 영역에 대한 감지기 인스턴스를 만들고 초기 작동 파라미터를 정의하는 데 사용됩니다.

CLI 명령은 다음을 사용합니다.

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

파일: 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}"
    }
  ]
}
```

응답:

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

이 예시에서 BatchPutMessage은(는) 영역 내 단일 센서에 대한 온도 센서 판독값을 보고하는 데 사용됩니다.

CLI 명령은 다음을 사용합니다.

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

파일: temperatureExample.json

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

응답:

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

이 예제에서 BatchPutMessage은(는) 특정 지역의 원하는 온도를 변경하는 데 사용됩니다.

CLI 명령은 다음을 사용합니다.

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

파일: seedSetDesiredTemp.json

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

```
}
```

응답:

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

Area51 감지기 인스턴스에서 생성된 Amazon SNS 메시지의 예:

```
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"
}
```

이 예시에서는 DescribeDetector API를 사용하여 감지기 인스턴스의 현재 상태에 대한 정보를 가져옵니다.

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

응답:

```
{
    "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"
    }
}
```

의 HVAC 시스템에 대한 BatchUpdateDetector 예제 AWS IoT Events

이 예시에서 `BatchUpdateDetector`은(는) 작동 중인 감지기 인스턴스의 작동 파라미터를 변경하는 데 사용됩니다.

효율적인 HVAC 시스템 관리에는 여러 탐지기에 대한 배치 업데이트가 필요한 경우가 많습니다. 이 섹션에서는 감지기에 AWS IoT Events의 배치 업데이트 기능을 사용하는 방법을 보여줍니다. 여러 제어 파라미터를 동시에 수정하고 임계값을 업데이트하여 디바이스 플릿에서 응답 작업을 조정하여 대규모 시스템을 효과적으로 관리하는 기능을 개선하는 방법을 알아봅니다.

CLI 명령은 다음을 사용합니다.

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

파일: `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": [
        ]
      }
    }
  ]
}
```

응답:

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

AWS IoT Core 규칙 엔진 및 AWS IoT Events

다음 규칙은 AWS IoT Events MQTT 메시지를 섀도우 업데이트 요청 메시지로 다시 게시합니다. 감지기 모델에 의해 제어되는 각 영역의 난방 장치 및 냉각 장치에 AWS IoT Core 사물이 정의되어 있다고 가정합니다.

이 예제에서는 이름이 `Area51HeatingUnit` 및 `Area51CoolingUnit`인 항목을 정의했습니다.

CLI 명령은 다음을 사용합니다.

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

파일: `ADMShadowCoolOffRule.json`

```
{
  "ruleName": "ADMShadowCoolOff",
  "topicRulePayload": {
    "sql": "SELECT topic(3) as state.desired.command FROM 'hvac/Cooling/Off'",
    "description": "areaDetectorModel mqtt topic publish to cooling unit shadow
 request",
    "ruleDisabled": false,
    "awsIotSqlVersion": "2016-03-23",
    "actions": [
      {
        "republish": {
          "topic": "$$aws/things/${payload.detector.keyValue}CoolingUnit/shadow/
update",
          "roleArn": "arn:aws:iam::123456789012:role/service-role/ADMShadowRole"
        }
      }
    ]
  }
}
```

응답: [비어 있음]

CLI 명령은 다음을 사용합니다.

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

파일: ADMShadowCoolOnRule.json

```
{
  "ruleName": "ADMShadowCoolOn",
  "topicRulePayload": {
    "sql": "SELECT topic(3) as state.desired.command FROM 'hvac/Cooling/On'",
    "description": "areaDetectorModel mqtt topic publish to cooling unit shadow
 request",
    "ruleDisabled": false,
    "awsIotSqlVersion": "2016-03-23",
    "actions": [
      {
        "republish": {
          "topic": "$$aws/things/${payload.detector.keyValue}CoolingUnit/shadow/
update",
          "roleArn": "arn:aws:iam::123456789012:role/service-role/ADMShadowRole"
        }
      }
    ]
  }
}
```

응답: [비어 있음]

CLI 명령은 다음을 사용합니다.

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

파일: 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"
        }
      }
    ]
  }
}
```

응답: [비어 있음]

CLI 명령은 다음을 사용합니다.

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

파일: 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"
        }
      }
    ]
  }
}
```

응답: [비어 있음]

예:를 사용하여 조건을 감지하는 크레인 AWS IoT Events

많은 크레인 운전자는 기계의 유지 보수 또는 교체가 필요한 시기를 감지하여 적절한 알림을 트리거하고자 합니다. 각 크레인에는 모터가 있습니다. 모터는 압력과 온도에 대한 정보가 포함된 메시지(입력)를 내보냅니다. 운영자는 두 가지 수준의 이벤트 감지기를 원합니다.

- 크레인 수준의 이벤트 감지기
- 모터 수준의 이벤트 감지기

운영자는 모터의 메시지(`craneId` 및 `motorid`가 모두 포함된 메타데이터 포함)를 사용하여 적절한 라우팅을 통해 두 수준의 이벤트 감지기를 모두 실행할 수 있습니다. 이벤트 조건이 충족되면 적절한 Amazon SNS 주제로 알림을 보내야 합니다. 운영자는 알림이 중복되지 않도록 감지기 모델을 구성할 수 있습니다.

이 예제는 다음과 같은 기능을 보여줍니다.

- 입력 생성, 읽기, 업데이트, 삭제(CRUD)
- 이벤트 감지기 모델 및 다양한 버전의 이벤트 감지기 생성, 읽기, 업데이트, 삭제(CRUD)
- 하나의 입력을 여러 이벤트 감지기로 라우팅합니다.
- 감지기 모델로의 입력
- 트리거 조건 및 라이프사이클 이벤트 평가.
- 조건의 상태 변수를 참조하고 조건에 따라 해당 값을 설정할 수 있습니다.
- 정의, 상태, 트리거 평가자 및 작업 실행자를 사용한 런타임 오케스트레이션.
- SNS 타겟을 사용한 `ActionsExecutor`의 작업 실행.

에서 감지된 조건에 대한 응답으로 명령 전송 AWS IoT Events

이 페이지에서는 AWS IoT Events 명령을 사용하여 입력을 설정하고, 감지기 모델을 생성하고, 시뮬레이션된 센서 데이터를 전송하는 예를 제공합니다. 이 예제에서는를 AWS IoT Events 사용하여 모터 및 크레인과 같은 산업 장비를 모니터링하는 방법을 보여줍니다.

```
#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
```

크레인 모니터링을 위한 AWS IoT Events 감지기 모델

장비 또는 디바이스 플릿의 장애 또는 작동 변경을 모니터링하고 이러한 이벤트가 발생할 때 작업을 트리거합니다. 상태, 규칙 및 작업을 지정하는 감지기 모델을 JSON으로 정의합니다. 이를 통해 온도 및 압력과 같은 입력을 모니터링하고, 임계값 위반을 추적하고, 알림을 보낼 수 있습니다. 이 예제는 크레인 및 모터에 대한 감지기 모델을 보여 주며, 과열 문제를 감지하고 임계값이 초과되면 Amazon SNS에서 알립니다. 모델을 업데이트하여 모니터링을 중단하지 않고 동작을 세분화할 수 있습니다.

파일: `craneDetectorModel.json`

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

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

```
}
```

기존 감지기 모델을 업데이트하려면. 파일: updateCraneDetectorModel.json

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

```
                                }
                            ]
                        },
                        {
                            "eventName": "Crane Threshold Breached",
                            "condition": "$variable.craneThresholdBreached > 5 &&
 $variable.alarmRaised == 'false'",
                            "actions": [
                                {
                                    "sns": {
                                        "targetArn": "arn:aws:sns:us-
east-1:123456789012:CraneSNSTopic"
                                    }
                                },
                                {
                                    "setVariable": {
                                        "variableName": "alarmRaised",
                                        "value": "'true'"
                                    }
                                }
                            ]
                        },
                        {
                            "eventName": "Underheated",
                            "condition": "$input.TemperatureInput.temperature < 10",
                            "actions": [
                                {
                                    "setVariable": {
                                        "variableName": "craneThresholdBreached",
                                        "value": "0"
                                    }
                                }
                            ]
                        }
                    ]
                }
            }
        ],
        "initialStateName": "Running"
    },
    "roleArn": "arn:aws:iam::123456789012:role/columboSNSRole"
}
```

파일: motorDetectorModel.json

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

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

기존 감지기 모델을 업데이트하려면. 파일: `updateMotorDetectorModel.json`

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

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

AWS IoT Events 크레인 모니터링을 위한 입력

이 예제에서는를 사용하여 크레인 모니터링 시스템에 대한 입력을 설정하는 방법을 보여줍니다 AWS IoT Events. 압력 및 온도 입력을 캡처하여 복잡한 산업 장비 모니터링을 위한 입력을 구성하는 방법을 보여줍니다.

파일: `pressureInput.json`

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

파일: temperatureInput.json

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

를 사용하여 경보 및 운영 메시지 전송 AWS IoT Events

크레인 모니터링 시스템에서는 효과적인 메시지 처리가 중요합니다. 이 섹션에서는 크레인 센서의 다양한 메시지 유형을 처리하고 이에 응답 AWS IoT Events 하도록를 구성하는 방법을 보여줍니다. 특정 메시지를 기반으로 경보를 설정하면 상태 업데이트를 구문 분석, 필터링 및 라우팅하여 적절한 작업을 트리거하는 데 도움이 될 수 있습니다.

파일: highPressureMessage.json

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

        }
```

```
    ]
}
```

파일: highTemperatureMessage.json

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

파일: lowPressureMessage.json

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

파일: lowTemperatureMessage.json

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

예: 센서 및 애플리케이션을 사용한 AWS IoT Events 이벤트 감지

이 감지기 모델은 AWS IoT Events 콘솔에서 사용할 수 있는 템플릿 중 하나입니다. 편의를 위해 여기에도 포함되어 있습니다.

이 예제는 센서 데이터를 사용한 AWS IoT Events의 애플리케이션 이벤트 감지를 보여줍니다. 적절한 작업을 트리거할 수 있도록 지정된 이벤트를 모니터링하는 감지기 모델을 생성하는 방법을 보여줍니다. 여러 센서 입력을 생성하고, 복잡한 이벤트 조건을 정의하고, 점진적 응답 메커니즘을 설정할 수 있습니다.

```
{
    "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"
    }
}
```

예: 와의 디바이스 연결을 모니터링하기 위한 디바이스 HeartBeat AWS IoT Events

이 감지기 모델은 AWS IoT Events 콘솔에서 사용할 수 있는 템플릿 중 하나입니다. 편의를 위해 여기에도 포함되어 있습니다.

Defective Heart Beat(DHB) 예제에서는 AWS IoT Events 를 의료 모니터링에 사용하는 방법을 보여줍니다. 이 예제에서는 심박수 데이터를 분석하고, 불규칙한 패턴을 감지하고, 적절한 응답을 트리거하는 감지기 모델을 생성하는 방법을 보여줍니다. 입력을 설정하고, 임계값을 정의하고, 잠재적 심장 문제에 대한 알림을 구성하여 관련 의료 애플리케이션에서 AWS IoT Events의 다양성을 보여주는 방법을 알아봅니다.

```
{
    "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"
    }
}
```

예:의 ISA 경보 AWS IoT Events

이 감지기 모델은 AWS IoT Events 콘솔에서 사용할 수 있는 템플릿 중 하나입니다. 편의를 위해 여기에도 포함되어 있습니다.

```
{
    "detectorModelName": "AWS_IoTEvents_Blueprints_ISA_Alarm",
    "detectorModelDefinition": {
        "states": [
            {
                "onInput": {
                    "transitionEvents": [
```

```
                        {
                            "eventName": "unshelve",
                            "actions": [],
                            "condition":
"$input.AWS_IoTEvents_Blueprints_ISA_Alarm_Input.command == \"unshelve\" &&
$variable.state == \"rtnunack\"",
                            "nextState": "RTN_Unacknowledged"
                        },
                        {
                            "eventName": "unshelve",
                            "actions": [],
                            "condition":
"$input.AWS_IoTEvents_Blueprints_ISA_Alarm_Input.command == \"unshelve\" &&
$variable.state == \"ack\"",
                            "nextState": "Acknowledged"
                        },
                        {
                            "eventName": "unshelve",
                            "actions": [],
                            "condition":
"$input.AWS_IoTEvents_Blueprints_ISA_Alarm_Input.command == \"unshelve\" &&
$variable.state == \"unack\"",
                            "nextState": "Unacknowledged"
                        },
                        {
                            "eventName": "unshelve",
                            "actions": [],
                            "condition":
"$input.AWS_IoTEvents_Blueprints_ISA_Alarm_Input.command == \"unshelve\" &&
$variable.state == \"normal\"",
                            "nextState": "Normal"
                        }
                    ],
                    "events": []
                },
                "stateName": "Shelved",
                "onEnter": {
                    "events": []
                },
                "onExit": {
                    "events": []
                }
            },
            {
```

```
                "onInput": {
                    "transitionEvents": [
                        {
                            "eventName": "abnormal_condition",
                            "actions": [],
                            "condition":
"$input.AWS_IoTEvents_Blueprints_ISA_Alarm_Input.value > $variable.higher_threshold ||
$input.AWS_IoTEvents_Blueprints_ISA_Alarm_Input.value < $variable.lower_threshold",
                            "nextState": "Unacknowledged"
                        },
                        {
                            "eventName": "acknowledge",
                            "actions": [],
                            "condition":
"$input.AWS_IoTEvents_Blueprints_ISA_Alarm_Input.command == \"acknowledge\"",
                            "nextState": "Normal"
                        },
                        {
                            "eventName": "shelve",
                            "actions": [],
                            "condition":
"$input.AWS_IoTEvents_Blueprints_ISA_Alarm_Input.command == \"shelve\"",
                            "nextState": "Shelved"
                        },
                        {
                            "eventName": "remove_from_service",
                            "actions": [],
                            "condition":
"$input.AWS_IoTEvents_Blueprints_ISA_Alarm_Input.command == \"remove\"",
                            "nextState": "Out_of_service"
                        },
                        {
                            "eventName": "suppression",
                            "actions": [],
                            "condition":
"$input.AWS_IoTEvents_Blueprints_ISA_Alarm_Input.command == \"suppressed\"",
                            "nextState": "Suppressed_by_design"
                        }
                    ],
                    "events": []
                },
                "stateName": "RTN_Unacknowledged",
                "onEnter": {
                    "events": [
```

```
                        {
                            "eventName": "State Save",
                            "actions": [
                                {
                                    "setVariable": {
                                        "variableName": "state",
                                        "value": "\"rtnunack\""
                                    }
                                }
                            ],
                            "condition": "true"
                        }
                    ]
                },
                "onExit": {
                    "events": []
                }
            },
            {
                "onInput": {
                    "transitionEvents": [
                        {
                            "eventName": "abnormal_condition",
                            "actions": [],
                            "condition":
"$input.AWS_IoTEvents_Blueprints_ISA_Alarm_Input.value > $variable.higher_threshold ||
$input.AWS_IoTEvents_Blueprints_ISA_Alarm_Input.value < $variable.lower_threshold",
                            "nextState": "Unacknowledged"
                        },
                        {
                            "eventName": "shelve",
                            "actions": [],
                            "condition":
"$input.AWS_IoTEvents_Blueprints_ISA_Alarm_Input.command == \"shelve\"",
                            "nextState": "Shelved"
                        },
                        {
                            "eventName": "remove_from_service",
                            "actions": [],
                            "condition":
"$input.AWS_IoTEvents_Blueprints_ISA_Alarm_Input.command == \"remove\"",
                            "nextState": "Out_of_service"
                        },
                        {
```

```
                            "eventName": "suppression",
                            "actions": [],
                            "condition":
"$input.AWS_IoTEvents_Blueprints_ISA_Alarm_Input.command == \"suppressed\"",
                            "nextState": "Suppressed_by_design"
                        }
                    ],
                    "events": [
                        {
                            "eventName": "Create Config variables",
                            "actions": [
                                {
                                    "setVariable": {
                                        "variableName": "lower_threshold",
                                        "value":
"$input.AWS_IoTEvents_Blueprints_ISA_Alarm_Input.lower_threshold"
                                    }
                                },
                                {
                                    "setVariable": {
                                        "variableName": "higher_threshold",
                                        "value":
"$input.AWS_IoTEvents_Blueprints_ISA_Alarm_Input.higher_threshold"
                                    }
                                }
                            ],
                            "condition": "$variable.lower_threshold !=
$variable.lower_threshold"
                        }
                    ]
                },
                "stateName": "Normal",
                "onEnter": {
                    "events": [
                        {
                            "eventName": "State Save",
                            "actions": [
                                {
                                    "setVariable": {
                                        "variableName": "state",
                                        "value": "\"normal\""
                                    }
                                }
                            ],
```

```
                            "condition": "true"
                        }
                    ]
                },
                "onExit": {
                    "events": []
                }
            },
            {
                "onInput": {
                    "transitionEvents": [
                        {
                            "eventName": "acknowledge",
                            "actions": [],
                            "condition":
"$input.AWS_IoTEvents_Blueprints_ISA_Alarm_Input.command == \"acknowledge\"",
                            "nextState": "Acknowledged"
                        },
                        {
                            "eventName": "return_to_normal",
                            "actions": [],
                            "condition":
"($input.AWS_IoTEvents_Blueprints_ISA_Alarm_Input.value <= $variable.higher_threshold
&& $input.AWS_IoTEvents_Blueprints_ISA_Alarm_Input.value >=
$variable.lower_threshold)",
                            "nextState": "RTN_Unacknowledged"
                        },
                        {
                            "eventName": "shelve",
                            "actions": [],
                            "condition":
"$input.AWS_IoTEvents_Blueprints_ISA_Alarm_Input.command == \"shelve\"",
                            "nextState": "Shelved"
                        },
                        {
                            "eventName": "remove_from_service",
                            "actions": [],
                            "condition":
"$input.AWS_IoTEvents_Blueprints_ISA_Alarm_Input.command == \"remove\"",
                            "nextState": "Out_of_service"
                        },
                        {
                            "eventName": "suppression",
                            "actions": [],
```

```
                            "condition":
"$input.AWS_IoTEvents_Blueprints_ISA_Alarm_Input.command == \"suppressed\"",
                            "nextState": "Suppressed_by_design"
                        }
                    ],
                    "events": []
                },
                "stateName": "Unacknowledged",
                "onEnter": {
                    "events": [
                        {
                            "eventName": "State Save",
                            "actions": [
                                {
                                    "setVariable": {
                                        "variableName": "state",
                                        "value": "\"unack\""
                                    }
                                }
                            ],
                            "condition": "true"
                        }
                    ]
                },
                "onExit": {
                    "events": []
                }
            },
            {
                "onInput": {
                    "transitionEvents": [
                        {
                            "eventName": "unsuppression",
                            "actions": [],
                            "condition":
"$input.AWS_IoTEvents_Blueprints_ISA_Alarm_Input.command == \"unsuppressed\" &&
$variable.state == \"normal\"",
                            "nextState": "Normal"
                        },
                        {
                            "eventName": "unsuppression",
                            "actions": [],
```

```
                            "condition":
"$input.AWS_IoTEvents_Blueprints_ISA_Alarm_Input.command == \"unsuppressed\" &&
$variable.state == \"unack\"",
                            "nextState": "Unacknowledged"
                        },
                        {
                            "eventName": "unsuppression",
                            "actions": [],
                            "condition":
"$input.AWS_IoTEvents_Blueprints_ISA_Alarm_Input.command == \"unsuppressed\" &&
$variable.state == \"ack\"",
                            "nextState": "Acknowledged"
                        },
                        {
                            "eventName": "unsuppression",
                            "actions": [],
                            "condition":
"$input.AWS_IoTEvents_Blueprints_ISA_Alarm_Input.command == \"unsuppressed\" &&
$variable.state == \"rtnunack\"",
                            "nextState": "RTN_Unacknowledged"
                        }
                    ],
                    "events": []
                },
                "stateName": "Suppressed_by_design",
                "onEnter": {
                    "events": []
                },
                "onExit": {
                    "events": []
                }
            },
            {
                "onInput": {
                    "transitionEvents": [
                        {
                            "eventName": "return_to_service",
                            "actions": [],
                            "condition":
"$input.AWS_IoTEvents_Blueprints_ISA_Alarm_Input.command == \"add\" && $variable.state
== \"rtnunack\"",
                            "nextState": "RTN_Unacknowledged"
                        },
                        {
```

```
                            "eventName": "return_to_service",
                            "actions": [],
                            "condition":
"$input.AWS_IoTEvents_Blueprints_ISA_Alarm_Input.command == \"add\" && $variable.state
== \"unack\"",
                            "nextState": "Unacknowledged"
                        },
                        {
                            "eventName": "return_to_service",
                            "actions": [],
                            "condition":
"$input.AWS_IoTEvents_Blueprints_ISA_Alarm_Input.command == \"add\" && $variable.state
== \"ack\"",
                            "nextState": "Acknowledged"
                        },
                        {
                            "eventName": "return_to_service",
                            "actions": [],
                            "condition":
"$input.AWS_IoTEvents_Blueprints_ISA_Alarm_Input.command == \"add\" && $variable.state
== \"normal\"",
                            "nextState": "Normal"
                        }
                    ],
                    "events": []
                },
                "stateName": "Out_of_service",
                "onEnter": {
                    "events": []
                },
                "onExit": {
                    "events": []
                }
            },
            {
                "onInput": {
                    "transitionEvents": [
                        {
                            "eventName": "re-alarm",
                            "actions": [],
                            "condition": "timeout(\"snooze\")",
                            "nextState": "Unacknowledged"
                        },
                        {
```

```
                            "eventName": "return_to_normal",
                            "actions": [],
                            "condition":
"$input.AWS_IoTEvents_Blueprints_ISA_Alarm_Input.command == \"reset\"",
                            "nextState": "Normal"
                        },
                        {
                            "eventName": "shelve",
                            "actions": [],
                            "condition":
"$input.AWS_IoTEvents_Blueprints_ISA_Alarm_Input.command == \"shelve\"",
                            "nextState": "Shelved"
                        },
                        {
                            "eventName": "remove_from_service",
                            "actions": [],
                            "condition":
"$input.AWS_IoTEvents_Blueprints_ISA_Alarm_Input.command == \"remove\"",
                            "nextState": "Out_of_service"
                        },
                        {
                            "eventName": "suppression",
                            "actions": [],
                            "condition":
"$input.AWS_IoTEvents_Blueprints_ISA_Alarm_Input.command == \"suppressed\"",
                            "nextState": "Suppressed_by_design"
                        }
                    ],
                    "events": []
                },
                "stateName": "Acknowledged",
                "onEnter": {
                    "events": [
                        {
                            "eventName": "Create Timer",
                            "actions": [
                                {
                                    "setTimer": {
                                        "seconds": 60,
                                        "timerName": "snooze"
                                    }
                                }
                            ],
                            "condition": "true"
```

```
                        },
                        {
                            "eventName": "State Save",
                            "actions": [
                                {
                                    "setVariable": {
                                        "variableName": "state",
                                        "value": "\"ack\""
                                    }
                                }
                            ],
                            "condition": "true"
                        }
                    ]
                },
                "onExit": {
                    "events": []
                }
            }
        ],
        "initialStateName": "Normal"
    },
    "detectorModelDescription": "This detector model is used to detect if a monitored
 device is in an Alarming State in accordance to the ISA 18.2.",
    "roleArn": "arn:aws:iam::123456789012:role/IoTEventsRole",
    "key": "alarmId"
}
```

예:를 사용하여 간단한 경보 빌드 AWS IoT Events

이 감지기 모델은 AWS IoT Events 콘솔에서 사용할 수 있는 템플릿 중 하나입니다. 편의를 위해 여기에도 포함되어 있습니다.

```
{
    "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"
}
```

에서 경보를 사용한 모니터링 AWS IoT Events

AWS IoT Events 경보는 데이터에 변경 사항이 있는지 모니터링하는 데 도움이 됩니다. 데이터는 디바이스 및 프로세스를 측정하는 지표가 될 수 있습니다. 임계값 위반 시 알림을 보내는 경보를 생성할 수 있습니다. 경보를 사용하면 문제를 감지하고, 유지보수를 간소화하고, 디바이스와 프로세스의 성능을 최적화할 수 있습니다.

경보는 경보 모델의 인스턴스입니다. 경보 모델은 감지할 대상, 전송 시기, 수신자 등을 지정합니다. 경보 상태가 변경될 때 발생하는 지원되는 작업을 하나 이상 지정할 수도 있습니다.는 데이터에서 파생된 입력 속성을 적절한 경보로 AWS IoT Events 라우팅합니다. 모니터링 중인 데이터가 지정된 범위를 벗어나면 경보가 호출됩니다. 경보를 확인하거나 스누즈 모드로 설정할 수도 있습니다.

작업 AWS IoT SiteWise

AWS IoT Events 경보를 사용하여의 자산 속성을 모니터링할 수 있습니다 AWS IoT SiteWise.는 자산 속성 값을 AWS IoT Events 경보로 AWS IoT SiteWise 전송합니다.는 경보 상태를 로 AWS IoT Events 전송합니다 AWS IoT SiteWise.

AWS IoT SiteWise 는 외부 경보도 지원합니다. 외부에서 경보를 사용하고 경보 상태 데이터를 반환하는 솔루션이 AWS IoT SiteWise 있는 경우 외부 경보를 선택할 수 있습니다. 외부 경보에는 경보 상태 데이터를 수집하는 측정 속성이 포함되어 있습니다.

AWS IoT SiteWise 는 외부 경보의 상태를 평가하지 않습니다. 또한 경보 상태가 변경될 때는 외부 경보를 승인하거나 일시 중지할 수 없습니다.

SiteWise 모니터 기능을 사용하여 SiteWise 모니터 포털에서 외부 경보의 상태를 볼 수 있습니다.

자세한 내용은 AWS IoT SiteWise 사용 설명서의 경보를 통한 데이터 모니터링 및 SiteWise Monitor 응용 프로그램 안내서의 경보를 통한 모니터링을 참조하십시오.

승인 이동

경보 모델을 만들 때 승인 이동의 활성화 여부를 선택합니다. 승인 이동을 활성화하면 경보 상태가 변경될 때 팀에 알림을 받습니다. 팀에서 경보를 확인하고 메모를 남길 수 있습니다. 예를 들어 경보 정보와 문제 해결을 위해 취할 조치를 포함할 수 있습니다. 모니터링 중인 데이터가 지정된 범위를 벗어나면 경보가 호출됩니다.

경보의 상태는 다음과 같습니다.

`DISABLED`

경보가 `DISABLED` 상태에 있으면 데이터를 평가할 준비가 되지 않은 것입니다. 경보를 활성화하려면 경보를 `NORMAL` 상태로 변경해야 합니다.

`NORMAL`

경보가 `NORMAL` 상태에 있으면 데이터를 평가할 준비가 된 것입니다.

`ACTIVE`

경보가 `ACTIVE` 상태에 있는 경우 경보가 호출됩니다. 모니터링 중인 데이터가 지정된 범위를 벗어났습니다.

`ACKNOWLEDGED`

경보가 `ACKNOWLEDGED` 상태에 있으면 경보가 호출되고 사용자가 경보를 확인한 것입니다.

`LATCHED`

경보가 호출되었지만 일정 시간이 지나도 경보를 확인하지 못했습니다. 경보는 자동으로 `NORMAL` 상태로 변경됩니다.

`SNOOZE_DISABLED`

경보가 `SNOOZE_DISABLED` 상태에 있으면 지정된 기간 동안 알람이 비활성화됩니다. 스누즈 시간이 지나면 경보가 자동으로 `NORMAL` 상태로 바뀝니다.

에서 경보 모델 생성 AWS IoT Events

AWS IoT Events 경보를 사용하여 데이터를 모니터링하고 임계값 위반 시 알림을 받을 수 있습니다. 경보는 경보 모델을 만들거나 구성하는 데 사용하는 파라미터를 제공합니다. AWS IoT Events 콘솔 또는 AWS IoT Events API를 사용하여 경보 모델을 생성하거나 구성할 수 있습니다. 경보 모델을 구성하면 새 데이터가 도착할 때 변경 사항이 적용됩니다.

요구 사항

경보 모델을 생성할 때는 다음 요구 사항이 적용됩니다.

- 경보 모델을 생성하여의 입력 속성 AWS IoT Events 또는의 자산 속성을 모니터링할 수 있습니다 AWS IoT SiteWise.
  - 경보 모델을 생성하기 AWS IoT Events에서 모델에 대한 입력 생성 AWS IoT Events 전에에서 입력 속성을 모니터링하도록 선택한 경우

- 자산 속성을 모니터링하도록 선택한 경우 경보 모델을 생성하기 전에에서 자산 모델을 생성해야 합니다. AWS IoT SiteWise
- 경보가 작업을 수행하고 AWS 리소스에 액세스할 수 있도록 허용하는 IAM 역할이 있어야 합니다. 자세한 내용은 AWS IoT Events에 권한 설정을 참조하십시오.
- 이 자습서에서 사용하는 모든 AWS 리소스는 동일한 AWS 리전에 있어야 합니다.

경보 모델 만들기(콘솔)

다음은 AWS IoT Events 콘솔에서 AWS IoT Events 속성을 모니터링하기 위해 경보 모델을 생성하는 방법을 보여줍니다.

1. AWS IoT Events 콘솔에 로그인합니다.
2. 탐색 창에서 경보 모델을 선택합니다.
3. 경보 모델 페이지에서 경보 모델 생성을 선택합니다.
4. 경보 모델 세부 정보 섹션에서 다음을 수행합니다.

   a. 고유한 이름을 입력합니다.

   b. (선택 사항) 설명을 입력합니다.
5. 경보 대상 섹션에서 다음을 수행합니다.

   > **Important**
   >
   > AWS IoT SiteWise 자산 속성을 선택한 경우 AWS IoT SiteWise에서 자산 모델을 생성해야 합니다.

   a. AWS IoT Events 입력 속성을 선택합니다.

   b. 입력을 선택합니다.

   c. 입력 속성 키를 선택합니다. 이 입력 속성은이 키와 연결된 입력을 경보로 AWS IoT Events 라우팅하는 경보를 생성하는 키로 사용됩니다.

   > **Important**
   >
   > 입력 메시지 페이로드에 이 입력 속성 키가 없거나 키에 지정된 동일한 JSON 경로에 키가 있지 않은 경우 메시지를 AWS IoT Events에서 수집할 수 없습니다.

6. 임계값 정의 섹션에서가 경보 상태를 변경하는 데 AWS IoT Events 사용하는 입력 속성, 임계값 및 비교 연산자를 정의합니다.

   a. 입력 속성에서 모니터링하려는 속성을 선택합니다.

      이 입력 속성이 새 데이터를 수신할 때마다 해당 입력 속성을 평가하여 경보 상태를 확인합니다.

   b. 연산자에서 비교 연산자를 선택합니다. 연산자는 입력 속성을 속성의 임계값과 비교합니다.

      다음 옵션 중에서 선택할 수 있습니다.

      - > 보다 큼
      - >= 보다 크거나 같은
      - < 보다 작음
      - <= 보다 작거나 같은
      - = 같음
      - != 같지 않음

   c. 임계값에 숫자를 입력하거나 AWS IoT Events input에서 속성을 선택합니다.는이 값을 선택한 입력 속성의 값과 AWS IoT Events 비교합니다.

   d. (선택 사항) 심각도에는 이 경보의 심각도를 반영하기 위해 팀에서 이해할 수 있는 숫자를 사용하십시오.

7. (선택 사항) 알림 설정 섹션에서 경보에 대한 알림 설정을 구성합니다.

   최대 10개의 알림을 추가할 수 있습니다. 알림 1에서 다음을 수행합니다.

   a. 프로토콜에서 다음 옵션 중에 선택합니다.

      - 이메일 및 문자 - SMS 알림과 이메일 알림으로 경보를 보냅니다.
      - 이메일 - 이메일 알림으로 경보를 보냅니다.
      - 텍스트 - SMS 알림으로 경보를 보냅니다.

   b. 발신자에서 이 경보에 대한 알림을 보낼 수 있는 이메일 주소를 지정합니다.

      발신자 목록에 이메일 주소를 더 추가하려면 발신자 추가를 선택합니다.

   c. (선택 사항) 수신자에서 수신자를 선택합니다.

      수신자 목록에 사용자를 더 추가하려면 새 사용자 추가를 선택합니다. 경보 모델에 새 사용자를 추가하려면 먼저 IAM Identity Center 스토어에 새 사용자를 추가해야 합니다. 자세한 내용

은 에서 경보 수신자의 IAM Identity Center 액세스 관리 AWS IoT Events 섹션을 참조하십시오.

d. (선택 사항) 추가 사용자 지정 메시지에서 경보가 탐지한 내용과 수신자가 취해야 하는 조치를 설명하는 메시지를 입력합니다.

8. 인스턴스 섹션에서 이 경보 모델을 기반으로 생성된 모든 경보 인스턴스를 활성화하거나 비활성화할 수 있습니다.

9. 고급 설정 섹션에서 다음을 수행합니다.

a. 승인 이동의 경우 알림을 활성화하거나 비활성화할 수 있습니다.

- 활성화를 선택한 경우 경보 상태가 변경될 때 알림을 받습니다. 경보 상태가 정상으로 돌아가기 전에 알림을 확인해야 합니다.
- 비활성화를 선택한 경우 별도의 조치가 필요하지 않습니다. 측정 값이 설정 범위로 돌아가면 경보가 자동으로 정상 상태로 변경됩니다.

자세한 내용은 승인 이동 섹션을 참조하십시오.

b. 권한에서 다음 옵션 중 하나를 선택합니다.

- AWS 정책 템플릿에서 새 역할을 생성하고 AWS IoT Events 자동으로 IAM 역할을 생성할 수 있습니다.
- 이 경보 모델이 작업을 수행하고 다른 AWS 리소스에 액세스할 수 있도록 허용하는 기존 IAM 역할을 사용할 수 있습니다.

자세한 내용은 AWS IoT Events의 자격 증명 및 액세스 관리를 참조하십시오.

c. 추가 알림 설정의 경우 AWS Lambda 함수를 편집하여 경보 알림을 관리할 수 있습니다. AWS Lambda 함수에 대해 다음 옵션 중 하나를 선택합니다.

- 새 AWS Lambda 함수 생성 -에서 새 AWS Lambda 함수를 AWS IoT Events 생성합니다.
- 기존 AWS Lambda 함수 사용 - AWS Lambda 함수 이름을 선택하여 기존 함수를 AWS Lambda 사용합니다.

가능한 작업에 대한 자세한 내용은 AWS IoT Events 다른 AWS 서비스 작업을 참조하십시오.

d. (선택 사항) 상태 설정 작업에서 경보 상태가 변경될 때 수행할 AWS IoT Events 작업을 하나 이상 추가할 수 있습니다.

10. (선택 사항) 경보를 관리하기 위해 태그를 추가할 수 있습니다. 자세한 내용은 AWS IoT Events 리소스에 태그 지정을 참조하십시오.
11. 생성(Create)을 선택합니다.

에서 경보에 응답 AWS IoT Events

경보에 효과적으로 대응하는 것은를 사용하여 IoT 시스템을 관리하는 데 중요한 측면입니다 AWS IoT Events. 알림 채널 설정, 에스컬레이션 절차 정의, 자동 응답 작업 구현 등 경보를 구성하고 처리하는 다양한 방법을 살펴봅니다. 미묘한 경보 조건을 생성하고, 알림의 우선순위를 지정하고, 다른 AWS 서비스와 통합하여 IoT 애플리케이션을 위한 대응형 경보 관리 시스템을 구축하는 방법을 알아봅니다.

승인 이동이 활성화되면 경보 상태가 변경될 때 알림을 받습니다. 경보에 대응하려면 경보를 확인, 비활성화, 활성화, 재설정 또는 일시 중지할 수 있습니다.

Console

다음은 AWS IoT Events 콘솔에서 경보에 대응하는 방법을 보여줍니다.

1. AWS IoT Events 콘솔에 로그인합니다.
2. 탐색 창에서 경보 모델을 선택합니다.
3. 대상 경보 모델을 선택합니다.
4. 경보 목록 섹션에서 대상 경보를 선택합니다.
5. 작업에서 다음 옵션 중 하나를 선택할 수 있습니다.

   - 승인 - 경보가 `ACKNOWLEDGED` 상태로 변경됩니다.
   - 비활성화 - 경보가 `DISABLED` 상태로 변경됩니다.
   - 활성화 - 경보가 `NORMAL` 상태로 변경됩니다.
   - 재설정 - 경보가 `NORMAL` 상태로 변경됩니다.
   - 스누즈 후 다음을 수행하십시오.

     1. 스누즈 길이를 선택하거나 사용자 지정 스누즈 길이를 입력합니다.
     2. 저장을 선택합니다.

     경보가 `SNOOZE_DISABLED` 상태로 바뀝니다

이러한 상태에 대한 자세한 내용은 승인 이동를 참조하십시오.

API

하나 이상의 경보에 응답하려면 다음 AWS IoT Events API 작업을 사용할 수 있습니다.

- BatchAcknowledgeAlarm
- BatchDisableAlarm
- BatchEnableAlarm
- BatchResetAlarm
- BatchSnoozeAlarm

에서 경보 알림 관리 AWS IoT Events

AWS IoT Events 는 Lambda와 통합되어 사용자 지정 이벤트 처리 기능을 제공합니다. 이 섹션에서는 AWS IoT Events 탐지기 모델 내에서 Lambda 함수를 사용하여 복잡한 로직을 실행하고, 외부 서비스와 상호 작용하고, 정교한 이벤트 처리를 구현할 수 있는 방법을 살펴봅니다.

AWS IoT Events 는 Lambda 함수를 사용하여 경보 알림을 관리합니다. 에서 제공하는 Lambda 함수를 사용하거나 새 함수를 AWS IoT Events 생성할 수 있습니다.

주제

- 에서 Lambda 함수 생성 AWS IoT Events
- 에서 제공하는 Lambda 함수 사용 AWS IoT Events
- 에서 경보 수신자의 IAM Identity Center 액세스 관리 AWS IoT Events

에서 Lambda 함수 생성 AWS IoT Events

AWS IoT Events 는 경보가 이메일 및 SMS 알림을 보내고 받을 수 있도록 하는 Lambda 함수를 제공합니다.

요구 사항

경보용 Lambda 함수를 만들려면 다음 요구 사항이 적용됩니다.

- 경보가 SMS 알림을 보내는 경우 Amazon SNS가 SMS 메시지를 전송하도록 구성되어 있는지 확인합니다.
  - 자세한 내용은 다음 설명서를 참조하세요.

  - Amazon Simple Notification Service 개발자 안내서의 Amazon SNS를 사용한 모바일 문자 메시징 및 Amazon SNS SMS 메시지의 발신 자격 증명.
  - AWS SMS 사용 설명서의 AWS End User Messaging SMS란 무엇입니까?
- 경보가 이메일 또는 SMS 알림을 보내는 경우에서 Amazon SES 및 Amazon SNS AWS Lambda 작업을 허용하는 IAM 역할이 있어야 합니다.

  예제 정책:

  JSON

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

```
            "logs:CreateLogGroup",
            "logs:CreateLogStream",
            "logs:PutLogEvents"
          ],
          "Resource" : "*"
        }
    ]
}
```

- AWS IoT Events 및 모두에 대해 동일한 AWS 리전을 선택해야 합니다 AWS Lambda. 지원되는 리전은 Amazon Web Services 일반 참조의 AWS IoT Events 엔드포인트 및 할당량과 AWS Lambda 엔드포인트 및 할당량을 참조하십시오.

를 AWS IoT Events 사용하기 위한 Lambda 함수 배포 CloudFormation

이 자습서에서는 CloudFormation 템플릿을 사용하여 Lambda 함수를 배포합니다. 이 템플릿은 Lambda 함수가 Amazon SES 및 Amazon SNS와 연동되도록 허용하는 IAM 역할을 자동으로 생성합니다.

다음은 AWS Command Line Interface (AWS CLI)를 사용하여 CloudFormation 스택을 생성하는 방법을 보여줍니다.

1. 디바이스의 터미널에서를 실행`aws --version`하여를 설치했는지 확인합니다 AWS CLI. 자세한 내용은 AWS Command Line Interface 사용 설명서의 AWS CLI최신 버전의 설치 또는 업데이트를 참조하세요.
2. 를 실행`aws configure list`하여이 자습서의 모든 AWS 리소스가 있는 AWS 리전 AWS CLI 에서를 구성했는지 확인합니다. 자세한 내용은 AWS Command Line Interface 사용 설명서의 명령을 사용하여 구성 설정 및 보기 섹션을 참조하세요.
3. CloudFormation 템플릿을 다운로드하십시오, notificationLambda.template.yaml.zip.

   > **Note**
   >
   > 파일을 다운로드하는 데 문제가 있는 경우 CloudFormation 템플릿에서도 템플릿을 사용할 수 있습니다.

4. 콘텐츠의 압축을 풀고 `notificationLambda.template.yaml`로 로컬로 저장합니다.
5. 디바이스에서 터미널을 열고 `notificationLambda.template.yaml` 파일을 다운로드한 디렉터리로 이동합니다.
6. CloudFormation 스택을 생성하려면 다음 명령을 실행합니다.

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

이 CloudFormation 템플릿을 수정하여 Lambda 함수와 그 동작을 사용자 지정할 수 있습니다.

> **Note**
>
> AWS Lambda 는 함수 오류를 두 번 재시도합니다. 함수가 모든 수신 요청을 처리할 만큼 용량이 충분하지 않은 경우 이벤트는 함수로 전송될 때까지 몇 시간 또는 며칠 동안 대기열에서 대기할 수 있습니다. 성공적으로 처리되지 않은 이벤트를 캡처하도록 함수에 대해 배달 못한 메시지 대기열(DLQ)을 구성할 수 있습니다. 자세한 정보는 AWS Lambda 개발자 안내서의 비동기 호출을 참조하십시오.

CloudFormation 콘솔에서 스택을 생성하거나 구성할 수도 있습니다. 자세한 내용을 알아보려면 AWS CloudFormation 사용 설명서의 스택 작업을 참조하십시오.

에 대한 사용자 지정 Lambda 함수 생성 AWS IoT Events

Lambda 함수를 생성하거나 AWS IoT Events에서 제공하는 함수를 수정할 수 있습니다.

사용자 지정 Lambda 함수를 만들려면 다음 요구 사항이 적용됩니다.

- Lambda 함수가 지정된 작업을 수행하고 AWS 리소스에 액세스할 수 있도록 허용하는 권한을 추가합니다.
- 에서 제공하는 Lambda 함수를 사용하는 경우 Python 3.7 런타임을 선택해야 AWS IoT Events합니다.

예제 Lambda 함수:

```
import boto3
import json
import logging
import datetime
logger = logging.getLogger()
logger.setLevel(logging.INFO)
ses = boto3.client('ses')
sns = boto3.client('sns')
```

```
def check_value(target):
  if target:
    return True
  return False

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

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

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

def lambda_handler(event, context):
  logging.info('Received event: ' + json.dumps(event))
  nep = json.loads(event.get('notificationEventPayload'))
```

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

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

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

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

자세한 정보는 AWS Lambda개발자 설명서의 AWS Lambda (이)란 무엇입니까?를 참조하십시오.

CloudFormation 템플릿

다음 CloudFormation 템플릿을 사용하여 Lambda 함수를 생성하십시오.

```
AWSTemplateFormatVersion: '2010-09-09'
Description: 'Notification Lambda for Alarm Model'
Resources:
  NotificationLambdaRole:
    Type: AWS::IAM::Role
    Properties:
      AssumeRolePolicyDocument:
        Statement:
          - Effect: Allow
            Principal:
              Service: lambda.amazonaws.com
            Action: sts:AssumeRole
      Path: "/"
      ManagedPolicyArns:
        - 'arn:aws:iam::aws:policy/AWSLambdaExecute'
      Policies:
        - PolicyName: "NotificationLambda"
          PolicyDocument:
            Version: "2012-10-17"
            Statement:
              - Effect: "Allow"
                Action:
                  - "ses:GetIdentityVerificationAttributes"
                  - "ses:SendEmail"
                  - "ses:VerifyEmailIdentity"
                Resource: "*"
              - Effect: "Allow"
                Action:
                  - "sns:Publish"
```

```
                - "sns:OptInPhoneNumber"
                - "sns:CheckIfPhoneNumberIsOptedOut"
                - "sms-voice:DescribeOptedOutNumbers"
              Resource: "*"
            - Effect: "Deny"
              Action:
                - "sns:Publish"
              Resource: "arn:aws:sns:*:*:*"
 NotificationLambdaFunction:
   Type: AWS::Lambda::Function
   Properties:
     Role: !GetAtt NotificationLambdaRole.Arn
     Runtime: python3.7
     Handler: index.lambda_handler
     Timeout: 300
     MemorySize: 3008
     Code:
       ZipFile: |
         import boto3
         import json
         import logging
         import datetime
         logger = logging.getLogger()
         logger.setLevel(logging.INFO)
         ses = boto3.client('ses')
         sns = boto3.client('sns')
         def check_value(target):
           if target:
             return True
           return False

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

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

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

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

          emails = event.get('emailConfigurations', [])
          logger.info('Start Sending Emails')
```

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

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

에서 제공하는 Lambda 함수 사용 AWS IoT Events

경보 알림을 사용하면에서 제공하는 Lambda 함수 AWS IoT Events 를 사용하여 경보 알림을 관리할 수 있습니다.

AWS IoT Events 에서 제공하는 Lambda 함수를 사용하여 경보 알림을 관리할 경우 다음 요구 사항이 적용됩니다.

- Amazon Simple Email Service(Amazon SES)에서 이메일 알림을 보내는 이메일 주소를 확인해야 합니다. 자세한 내용은 Amazon Simple Email Service 개발자 가이드의 이메일 주소 자격 증명 확인을 참조하세요.

  확인 링크를 받은 경우 링크를 클릭하여 이메일 주소를 확인하십시오. 스팸 폴더에 이메일이 있는지 확인할 수도 있습니다.
- SMS 알림으로 경보를 보내는 경우 전화번호에는 E.164 국제 전화번호 형식을 사용해야 합니다. 이 형식에는 `+<country-calling-code><area-code><phone-number>`가 포함됩니다.

  전화번호 예시:

국가	지역 전화번호	E.164 형식 숫자
미국	206-555-0100	+12065550100
영국	020-1234-1234	+442012341234
리투아니아	8+601+12345	+37060112345

  국가 전화 코드는 countrycode.org에서 찾으십시오.

  에서 제공하는 Lambda 함수는 E.164 형식의 전화번호를 사용하는지 AWS IoT Events 확인합니다. 하지만 전화번호를 확인하지는 않습니다. 전화번호를 정확하게 입력했지만 SMS 알림을 받지 못한 경우 이동통신사에 문의할 수 있습니다. 이동통신사에서 메시지를 차단할 수 있습니다.

에서 경보 수신자의 IAM Identity Center 액세스 관리 AWS IoT Events

AWS IoT Events 는 AWS IAM Identity Center 를 사용하여 경보 수신자의 SSO 액세스를 관리합니다. AWS IoT Events 알림 수신자를 위해 IAM Identity Center를 구현하면 보안 및 사용자 경험을 개선할 수 있습니다. 경보가 수신자에게 알림을 보낼 수 있게 하려면 IAM Identity Center를 활성화하고 IAM ID 센터 스토어에 수신자를 추가해야 합니다. 자세한 내용은 AWS IAM Identity Center 사용 설명서의 사용자 추가를 참조하십시오.

> **Important**
>
> - AWS IoT Events AWS Lambda및 IAM Identity Center에 대해 동일한 AWS 리전을 선택해야 합니다.

- AWS Organizations는 한 번에 하나의 IAM Identity Center 리전만 지원합니다. IAM Identity Center를 다른 리전에서 사용할 수 있게 하려면 먼저 현재 IAM Identity Center 구성을 삭제해야 합니다. 자세한 내용은 AWS IAM Identity Center 사용 설명서의 IAM Identity Center 리전 Data를 참조하십시오.

의 보안 AWS IoT Events

의 클라우드 보안 AWS 이 최우선 순위입니다. AWS 고객은 보안에 가장 민감한 조직의 요구 사항을 충족하도록 구축된 데이터 센터 및 네트워크 아키텍처의 이점을 누릴 수 있습니다.

보안은 AWS 와 사용자 간의 공동 책임입니다. 공동 책임 모델은 이를 클라우드의 보안과 클라우드 내 보안으로 설명합니다.

- 클라우드 보안 - AWS 는 AWS 클라우드에서 AWS 서비스를 실행하는 인프라를 보호할 책임이 있습니다. AWS 또한는 안전하게 사용할 수 있는 서비스를 제공합니다. 서드 파티 감사자는 정기적으로 AWS 규정 준수 프로그램의 일환으로 보안 효과를 테스트하고 검증합니다. 에 적용되는 규정 준수 프로그램에 대한 자세한 내용은 AWS 규정 준수 프로그램별 범위 내 서비스를 AWS IoT Events참조하세요.
- 클라우드의 보안 - 사용자의 책임은 사용하는 AWS 서비스에 따라 결정됩니다. 또한 데이터의 민감도, 조직의 요구 사항, 관련 법률 및 규정을 비롯한 기타 요소에 대해서도 책임이 있습니다.

이 설명서는를 사용할 때 공동 책임 모델을 적용하는 방법을 이해하는 데 도움이 됩니다 AWS IoT Events. 다음 주제에서는 보안 및 규정 준수 목표를 충족하도록 AWS IoT Events 를 구성하는 방법을 보여줍니다. 또한 AWS IoT Events 리소스를 모니터링하고 보호하는 데 도움이 되는 다른 AWS 서비스를 사용하는 방법도 알아봅니다.

주제

- 에 대한 자격 증명 및 액세스 관리 AWS IoT Events
- 안정성, 가용성 및 성능을 유지하기 AWS IoT Events 위한 모니터링
- 에 대한 규정 준수 검증 AWS IoT Events
- 의 복원력 AWS IoT Events
- 의 인프라 보안 AWS IoT Events

에 대한 자격 증명 및 액세스 관리 AWS IoT Events

AWS Identity and Access Management (IAM)는 관리자가 AWS 리소스에 대한 액세스를 안전하게 제어할 수 있도록 지원하는 AWS 서비스입니다. IAM 관리자는 누가 AWS IoT Events 리소스를 사용할 수 있는 인증(로그인) 및 권한(권한 있음)을 받을 수 있는지 제어합니다. IAM은 추가 비용 없이 사용할 수 있는 AWS 서비스입니다.

주제

대상

AWS Identity and Access Management (IAM)를 사용하는 방법은 역할에 따라 다릅니다.

- 서비스 사용자 - 기능에 액세스할 수 없는 경우 관리자에게 권한 요청(참조AWS IoT Events 자격 증명 및 액세스 문제 해결)
- 서비스 관리자 - 사용자 액세스 결정 및 권한 요청 제출(AWS IoT Events 에서 IAM을 사용하는 방법 참조)
- IAM 관리자 - 액세스를 관리하기 위한 정책 작성(AWS IoT Events 자격 증명 기반 정책 예제 참조)

ID를 통한 인증

인증은 자격 증명 자격 증명을 AWS 사용하여에 로그인하는 방법입니다. AWS 계정 루트 사용자, IAM 사용자 또는 IAM 역할을 수임하여 인증되어야 합니다.

AWS IAM Identity Center (IAM Identity Center), Single Sign-On 인증 또는 Google/Facebook 자격 증명과 같은 자격 증명 소스의 자격 증명을 사용하여 페더레이션 자격 증명으로 로그인할 수 있습니다. 로그인하는 방법에 대한 자세한 내용은 AWS Sign-In 사용 설명서의 AWS 계정에 로그인하는 방법 섹션을 참조하세요.

프로그래밍 방식 액세스를 위해는 요청에 암호화 방식으로 서명할 수 있는 SDK 및 CLI를 AWS 제공합니다. 자세한 내용은 IAM 사용 설명서의 API 요청용AWS Signature Version 4 섹션을 참조하세요.

AWS 계정 루트 사용자

를 생성할 때 모든 AWS 서비스 및 리소스에 대한 완전한 액세스 권한이 있는 AWS 계정 theroot 사용자라는 하나의 로그인 자격 증명으로 AWS 계정시작합니다. 일상적인 태스크에 루트 사용자를 사용하지 않을 것을 강력히 권장합니다. 루트 사용자가 필요한 작업 목록은 IAM 사용자 설명서의 루트 사용자 자격 증명이 필요한 작업을 참조하세요.

IAM 사용자 및 그룹

IAM 사용자는 단일 개인 또는 애플리케이션에 대한 특정 권한을 가진 ID입니다. 장기 자격 증명이 있는 IAM 사용자 대신 임시 자격 증명을 사용하는 것이 좋습니다. 자세한 내용은 IAM 사용 설명서의 자격 증명 공급자와의 페더레이션을 사용하여 임시 자격 증명을 AWS 사용하여에 액세스하도록 인간 사용자에게 요구하기를 참조하세요.

IAM 그룹은 IAM 사용자 모음을 지정하고 대규모 사용자 집합에 대한 관리 권한을 더 쉽게 만듭니다. 자세한 내용은 IAM 사용 설명서의 IAM 사용자 사용 사례 섹션을 참조하세요.

IAM 역할

IAM 역할은 임시 자격 증명을 제공하는 특정 권한이 있는 자격 증명입니다. 사용자에서 IAM 역할(콘솔)로 전환하거나 또는 API 작업을 호출하여 역할을 수임할 수 있습니다. AWS CLI AWS 자세한 내용은 IAM 사용 설명서의 역할 수임 방법을 참조하세요.

IAM 역할은 페더레이션 사용자 액세스, 임시 IAM 사용자 권한, 교차 계정 액세스, 교차 서비스 액세스 및 Amazon EC2에서 실행되는 애플리케이션에 유용합니다. 자세한 내용은 IAM 사용 설명서의 교차 계정 리소스 액세스를 참조하세요.

정책을 사용하여 액세스 관리

정책을 AWS 생성하고 자격 증명 또는 리소스에 연결하여 AWS 에서 액세스를 제어합니다. 정책은 자격 증명 또는 리소스와 연결될 때 권한을 정의합니다.는 보안 주체가 요청할 때 이러한 정책을 AWS 평가합니다. 대부분의 정책은에 JSON 문서 AWS 로 저장됩니다. JSON 정책 문서에 대한 자세한 내용은 IAM 사용 설명서의 JSON 정책 개요 섹션을 참조하세요.

정책을 사용하여 관리자는 어떤 보안 주체가 어떤 리소스에 대해 어떤 조건에서 작업을 수행할 수 있는지 정의하여 누가 무엇을 액세스할 수 있는지 지정합니다.

기본적으로 사용자 및 역할에는 어떠한 권한도 없습니다. IAM 관리자는 IAM 정책을 생성하고 사용자가 수임할 수 있는 역할에 추가합니다. IAM 정책은 작업을 수행하기 위해 사용하는 방법과 관계없이 작업에 대한 권한을 정의합니다.

ID 기반 정책

ID 기반 정책은 ID(사용자, 사용자 그룹 또는 역할)에 연결하는 JSON 권한 정책 문서입니다. 이러한 정책은 자격 증명이 수행할 수 있는 작업, 대상 리소스 및 이에 관한 조건을 제어합니다. ID 기반 정책을 생성하는 방법을 알아보려면 IAM 사용 설명서에서 고객 관리형 정책으로 사용자 지정 IAM 권한 정의를 참조하세요.

ID 기반 정책은 인라인 정책(단일 ID에 직접 포함) 또는 관리형 정책(여러 ID에 연결된 독립 실행형 정책)일 수 있습니다. 관리형 정책 또는 인라인 정책을 선택하는 방법을 알아보려면 IAM 사용 설명서의 관리형 정책 및 인라인 정책 중에서 선택 섹션을 참조하세요.

기타 정책 유형

AWS 는 보다 일반적인 정책 유형에서 부여한 최대 권한을 설정할 수 있는 추가 정책 유형을 지원합니다.

- 권한 경계 - ID 기반 정책에서 IAM 엔터티에 부여할 수 있는 최대 권한을 설정합니다. 자세한 정보는 IAM 사용 설명서의 IAM 엔터티의 권한 범위를 참조하세요.
- 서비스 제어 정책(SCP) - AWS Organizations내 조직 또는 조직 단위에 대한 최대 권한을 지정합니다. 자세한 내용은AWS Organizations 사용 설명서의 서비스 제어 정책을 참조하세요.
- 리소스 제어 정책(RCP) – 계정의 리소스에 사용할 수 있는 최대 권한을 설정합니다. 자세한 내용은 AWS Organizations 사용 설명서의 리소스 제어 정책(RCP)을 참조하세요.
- 세션 정책 – 역할 또는 페더레이션 사용자에 대해 임시 세션을 프로그래밍 방식으로 생성할 때 파라미터로 전달하는 고급 정책입니다. 자세한 내용은 IAM 사용 설명서의 세션 정책을 참조하세요.

여러 정책 유형

여러 정책 유형이 요청에 적용되는 경우, 결과 권한은 이해하기가 더 복잡합니다. 에서 여러 정책 유형이 관련될 때 요청을 허용할지 여부를 AWS 결정하는 방법을 알아보려면 IAM 사용 설명서의 정책 평가 로직을 참조하세요.

자격 증명 및 액세스 관리에 대한 추가 정보

의 자격 증명 및 액세스 관리에 대한 자세한 내용은 다음 페이지를 AWS IoT Events참조하십시오.

- AWS IoT Events 에서 IAM을 사용하는 방법
- AWS IoT Events 자격 증명 및 액세스 문제 해결

AWS IoT Events 에서 IAM을 사용하는 방법

IAM을 사용하여 액세스를 관리하기 전에 사용할 수 있는 IAM 기능을 이해해야 AWS IoT Events합니다 AWS IoT Events. AWS IoT Events 및 기타 AWS 서비스에서 IAM을 사용하는 방법을 개괄적으로 알아보려면 IAM 사용 설명서의 AWS IAM으로 작업하는 서비스를 참조하세요.

주제

- AWS IoT Events 자격 증명 기반 정책
- AWS IoT Events 리소스 기반 정책
- AWS IoT Events 태그 기반 권한 부여
- AWS IoT Events IAM 역할

AWS IoT Events 자격 증명 기반 정책

IAM ID 기반 정책을 사용하면 허용되거나 거부되는 작업과 리소스 및 작업이 허용되거나 거부되는 조건을 지정할 수 있습니다. AWS IoT Events 는 특정 작업, 리소스 및 조건 키를 지원합니다. JSON 정책에서 사용하는 모든 요소에 대해 알아보려면 IAM 사용 설명서의 IAM JSON 정책 요소 참조를 참조하세요.

작업

IAM 자격 증명 기반 정책의 `Action` 요소는 정책에 따라 허용되거나 거부되는 특정 작업에 대해 설명합니다. 정책 작업은 일반적으로 연결된 AWS API 작업과 이름이 동일합니다. 이 작업은 연결된 작업을 수행할 수 있는 권한을 부여하기 위한 정책에서 사용됩니다.

의 정책 작업은 작업 앞에 접두사를 AWS IoT Events 사용합니다`iotevents:`. 예를 들어 API 작업으로 AWS IoT Events 입력을 생성할 수 있는 AWS IoT Events `CreateInput` 권한을 부여하려면 해당 정책에 `iotevents:CreateInput` 작업을 포함합니다. API 작업으로 입력을 전송할 수 있는 AWS IoT Events `BatchPutMessage` 권한을 부여하려면 해당 정책에 `iotevents-data:BatchPutMessage` 작업을 포함합니다. 정책 설명에는 `Action` 또는 `NotAction` 요소가 포함되어야 합니다.는이 서비스로 수행할 수 있는 작업을 설명하는 고유한 작업 세트를 AWS IoT Events 정의합니다.

명령문 하나에 여러 태스크를 지정하려면 다음과 같이 쉼표로 구분합니다.

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

와일드카드(*)를 사용하여 여러 작업을 지정할 수 있습니다. 예를 들어, Describe라는 단어로 시작하는 모든 작업을 지정하려면 다음 작업을 포함합니다.

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

AWS IoT Events 작업 목록을 보려면 IAM 사용 설명서의 에서 정의한 작업을 AWS IoT Events 참조하세요.

리소스

Resource 요소는 작업이 적용되는 객체를 지정합니다. 문에는 Resource또는 NotResource요소가 반드시 추가되어야 합니다. ARN을 사용하거나 문이 모든 리소스에 적용됨을 표시하는 와일드카드(*)를 사용하여 리소스를 지정합니다.

AWS IoT Events 감지기 모델 리소스의 ARN은 다음과 같습니다.

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

ARN 형식에 대한 자세한 내용은 Amazon AWS 리소스 이름(ARNs)을 사용하여 리소스 식별을 참조하세요.

예를 들어, 명령문에 Foobar 감지기 모델을 지정하려면 다음 ARN을 사용합니다.

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

특정 계정에 속하는 모든 인스턴스를 지정하려면 와일드카드(*)를 사용합니다.

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

리소스를 생성하기 위한 작업과 같은 일부 AWS IoT Events 작업은 특정 리소스에서 수행할 수 없습니다. 이러한 경우, 와일드카드(*)를 사용해야 합니다.

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

일부 AWS IoT Events API 작업에는 여러 리소스가 포함됩니다. 예를 들어 CreateDetectorModel은(는) 명령문의 입력을 참조하므로, 사용자에게 입력 사용 권한과 감지기 모델 사용 권한이 있어야 합니다. 단일 문에서 여러 리소스를 지정하려면 ARN을 쉼표로 구분합니다.

```
"Resource": [
```

```
        "resource1",
        "resource2"
```

AWS IoT Events 리소스 유형 및 해당 ARNs 목록을 보려면 IAM 사용 설명서의 에서 정의한 리소스를 AWS IoT Events 참조하세요. 각 리소스의 ARN을 지정할 수 있는 작업을 알아보려면 AWS IoT Events 가 정의한 작업을 참조하세요.

조건 키

Condition 요소(또는 Condition 블록)를 사용하면 정책이 발효되는 조건을 지정할 수 있습니다. Condition 요소는 옵션입니다. 같음, 미만 등 조건 연산자를 사용하여 정책의 조건을 요청의 값과 일치시키는 조건식을 빌드할 수 있습니다.

한 문에서 여러 Condition 요소를 지정하거나 단일 Condition 요소에서 여러 키를 지정하는 경우, AWS 는 논리적 AND 작업을 사용하여 평가합니다. 단일 조건 키에 여러 값을 지정하는 경우는 논리적 OR 작업을 사용하여 조건을 AWS 평가합니다. 문의 권한을 부여하기 전에 모든 조건을 충족해야 합니다.

조건을 지정할 때 자리 표시자 변수를 사용할 수도 있습니다. 예를 들어, 사용자에게 사용자 이름으로 태그가 지정된 경우에만 리소스에 액세스할 수 있는 권한을 부여할 수 있습니다. 자세한 내용은 IAM 사용 설명서의 IAM 정책 요소: 변수 및 태그 섹션을 참조하십시오.

AWS IoT Events 는 서비스별 조건 키를 제공하지 않지만 일부 전역 조건 키 사용을 지원합니다. 모든 AWS 전역 조건 키를 보려면 IAM 사용 설명서의 AWS 전역 조건 컨텍스트 키를 참조하세요."

예제

자격 AWS IoT Events 증명 기반 정책의 예를 보려면 섹션을 참조하세요AWS IoT Events 자격 증명 기반 정책 예제.

AWS IoT Events 리소스 기반 정책

AWS IoT Events 는 리소스 기반 정책을 지원하지 않습니다." 자세한 리소스 기반 정책 페이지의 예를 보려면 https://docs.aws.amazon.com/lambda/latest/dg/access-control-resource-based.html 단원을 참조하십시오.

AWS IoT Events 태그 기반 권한 부여

AWS IoT Events 리소스에 태그를 연결하거나 요청에 태그를 전달할 수 있습니다 AWS IoT Events. 태그에 근거하여 액세스를 제어하려면 iotevents:ResourceTag/*key-name*,

aws:RequestTag/*key-name*또는 aws:TagKeys조건 키를 사용하여 정책의 조건 요소에 태그 정보를 제공합니다. AWS IoT Events 리소스 태깅에 대한 자세한 내용은 AWS IoT Events 리소스에 태그 지정을 참조하세요.

리소스의 태그를 기반으로 리소스에 대한 액세스를 제한하는 자격 증명 기반 정책의 예시는 태그를 기반으로 AWS IoT Events 입력 보기에서 확인할 수 있습니다.

AWS IoT Events IAM 역할

IAM 역할은 특정 권한이 AWS 계정 있는 내의 엔터티입니다.

에서 임시 자격 증명 사용 AWS IoT Events

임시 보안 인증을 사용하여 페더레이션을 통해 로그인하거나, IAM 역할을 맡거나, 교차 계정 역할을 맡을 수 있습니다. AssumeRole 또는 GetFederationToken과 같은 AWS Security Token Service (AWS STS) API 작업을 호출하여 임시 보안 자격 증명을 얻습니다.

AWS IoT Events 는 임시 자격 증명 사용을 지원하지 않습니다.

서비스 연결 역할

서비스 연결 역할을 사용하면 AWS 서비스가 다른 서비스의 리소스에 액세스하여 사용자를 대신하여 작업을 완료할 수 있습니다. 서비스 연결 역할은 IAM 계정에 나타나고 서비스가 소유합니다. IAM 관리자는 서비스 연결 역할의 권한을 볼 수 있지만 편집할 수 없습니다.

AWS IoT Events 는 서비스 연결 역할을 지원하지 않습니다.

서비스 역할

이 기능을 사용하면 서비스가 사용자를 대신하여 서비스 역할을 수임할 수 있습니다. 이 역할을 사용하면 서비스가 다른 서비스의 리소스에 액세스해 사용자를 대신해 작업을 완료할 수 있습니다. 서비스 역할은 IAM 계정에 나타나고, 해당 계정이 소유합니다. 즉, IAM 관리자가 이 역할에 대한 권한을 변경할 수 있습니다. 그러나 권한을 변경하면 서비스의 기능이 손상될 수 있습니다.

AWS IoT Events 는 서비스 역할을 지원합니다.

AWS IoT Events 자격 증명 기반 정책 예제

기본적으로 사용자 및 역할에는 AWS IoT Events 리소스를 생성하거나 수정할 수 있는 권한이 없습니다. 또한 AWS Management Console AWS CLI또는 AWS API를 사용하여 작업을 수행할 수 없습니다.

IAM 관리자는 지정된 리소스에서 특정 API 작업을 수행할 수 있는 권한을 사용자와 역할에게 부여하는 IAM 정책을 생성해야 합니다. 그런 다음 관리자는 해당 권한이 필요한 사용자 또는 그룹에 이러한 정책을 연결해야 합니다.

이러한 예제 JSON 정책 문서를 사용하여 IAM 자격 증명 기반 정책을 생성하는 방법을 알아보려면 IAM 사용 설명서의 JSON 탭에서 정책 생성을 참조하세요.

주제

- 정책 모범 사례
- AWS IoT Events 콘솔 사용
- 사용자가에서 자신의 권한을 볼 수 있도록 허용 AWS IoT Events
- 하나의 AWS IoT Events 입력에 액세스
- 태그를 기반으로 AWS IoT Events 입력 보기

정책 모범 사례

자격 증명 기반 정책은 매우 강력합니다. 계정에서 사용자가 AWS IoT Events 리소스를 생성, 액세스 또는 삭제할 수 있는지 여부를 결정합니다. 이 작업으로 인해 AWS 계정에 비용이 발생할 수 있습니다. ID 기반 정책을 생성하거나 편집할 때는 다음 지침과 권장 사항을 따르세요.

- AWS 관리형 정책 사용 시작하기 - AWS IoT Events 빠르게 사용을 시작하려면 AWS 관리형 정책을 사용하여 직원에게 필요한 권한을 부여하세요. 이 정책은 이미 계정에서 사용할 수 있으며 AWS에 의해 유지 관리 및 업데이트됩니다. 자세한 내용은 IAM 사용 설명서의 AWS 관리형 정책으로 권한 사용 시작하기를 참조하세요.
- 최소 권한 부여 – 사용자 지정 정책을 생성할 때는 작업을 수행하는 데 필요한 권한만 부여합니다. 최소한의 권한 조합으로 시작하여 필요에 따라 추가 권한을 부여합니다. 처음부터 권한을 많이 부여한 후 나중에 줄이는 방법보다 이 방법이 안전합니다. 자세한 정보는 IAM 사용 설명서의 최소 권한 부여를 참조하십시오.
- 민감한 작업에 대해 MFA 활성화 – 보안을 강화하기 위해 사용자가 중요한 리소스 또는 API 작업에 액세스하려면 다중 인증(MFA)을 사용해야 합니다. 자세한 정보는 IAM 사용 설명서의 AWS에서 다중 인증(MFA) 사용을 참조하십시오.
- 보안 강화를 위해 정책 조건 사용 – 실제로 가능한 경우, 자격 증명 기반 정책이 리소스에 대한 액세스를 허용하는 조건을 정의합니다. 예를 들어, 요청을 할 수 있는 IP 주소의 범위를 지정하도록 조건을 작성할 수 있습니다. 지정된 날짜 또는 시간 범위 내에서만 요청을 허용하거나, SSL 또는 MFA를 사용해야 하는 조건을 작성할 수도 있습니다. 자세한 내용은 IAM 사용 설명서의 IAM JSON 정책 요소: 조건을 참조하십시오.

AWS IoT Events 콘솔 사용

AWS IoT Events 콘솔에 액세스하려면 최소 권한 집합이 있어야 합니다. 이러한 권한은의 AWS IoT Events 리소스에 대한 세부 정보를 나열하고 볼 수 있도록 허용해야 합니다 AWS 계정. 최소 필수 권한보다 더 제한적인 ID 기반 정책을 생성하는 경우, 콘솔이 해당 정책에 연결된 엔티티(사용자 또는 역할)에 대해 의도대로 작동하지 않습니다.

이러한 엔터티가 AWS IoT Events 콘솔을 계속 사용할 수 있도록 하려면 다음 AWS 관리형 정책도 엔터티에 연결합니다. 자세한 내용은 IAM 사용 설명서의 사용자에게 권한 추가를 참조하십시오.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iotevents:BatchPutMessage",
                "iotevents:BatchUpdateDetector",
                "iotevents:CreateDetectorModel",
                "iotevents:CreateInput",
                "iotevents:DeleteDetectorModel",
                "iotevents:DeleteInput",
                "iotevents:DescribeDetector",
                "iotevents:DescribeDetectorModel",
                "iotevents:DescribeInput",
                "iotevents:DescribeLoggingOptions",
                "iotevents:ListDetectorModelVersions",
                "iotevents:ListDetectorModels",
                "iotevents:ListDetectors",
                "iotevents:ListInputs",
                "iotevents:ListTagsForResource",
                "iotevents:PutLoggingOptions",
                "iotevents:TagResource",
                "iotevents:UntagResource",
                "iotevents:UpdateDetectorModel",
                "iotevents:UpdateInput",
                "iotevents:UpdateInputRouting"
            ],
            "Resource": "arn:aws:iotevents:us-
east-1:123456789012:detectorModel/your-detector-model-name",
```

```
    "Resource": "arn:aws:iotevents:us-east-1:123456789012:input/your-input-name"
        }
    ]
}
```

AWS CLI 또는 AWS API만 호출하는 사용자에게는 최소 콘솔 권한을 허용할 필요가 없습니다. 그 대신 수행하려는 API 작업과 일치하는 작업에만 액세스할 수 있도록 합니다.

사용자가에서 자신의 권한을 볼 수 있도록 허용 AWS IoT Events

이 예제는 사용자가 자신의 사용자 자격 증명에 연결된 인라인 및 관리형 정책을 볼 수 있도록 허용하는 정책을 생성하는 방법을 보여 줍니다. 사용자가 자신의 IAM 권한을 볼 수 있도록 허용하면 보안 인식 및 셀프 서비스 기능에 유용합니다. 이 정책에는 콘솔에서 또는 AWS CLI 또는 AWS API를 사용하여 프로그래밍 방식으로이 작업을 완료할 수 있는 권한이 포함됩니다.

JSON

```
{
       "Version":"2012-10-17",
       "Statement": [
           {
               "Sid": "ViewOwnUserInfo",
               "Effect": "Allow",
               "Action": [
                   "iam:GetUserPolicy",
                   "iam:ListGroupsForUser",
                   "iam:ListAttachedUserPolicies",
                   "iam:ListUserPolicies",
                   "iam:GetUser"
               ],
               "Resource": [
                   "arn:aws:iam::*:user/${aws:username}"
               ]
           },
           {
               "Sid": "NavigateInConsole",
               "Effect": "Allow",
               "Action": [
                   "iam:GetGroupPolicy",
                   "iam:GetPolicyVersion",
                   "iam:GetPolicy",
```

```
                "iam:ListAttachedGroupPolicies",
                "iam:ListGroupPolicies",
                "iam:ListPolicyVersions",
                "iam:ListPolicies",
                "iam:ListUsers"
            ],
            "Resource": "*"
        }
    ]
}
```

하나의 AWS IoT Events 입력에 액세스

다중 사용자 또는 다중 팀 환경에서 보안을 유지하려면 AWS IoT Events 입력에 대한 세분화된 액세스 제어가 중요합니다. 이 섹션에서는 특정 AWS IoT Events 입력에 대한 액세스 권한을 부여하는 동시에 다른 입력에 대한 액세스를 제한하는 IAM 정책을 생성하는 방법을 보여줍니다.

이 예제에서는의 사용자에게 AWS IoT Events 입력 중 하나인에 대한 AWS 계정 액세스 권한을 부여할 수 있습니다`exampleInput`. 사용자가 입력을 추가, 업데이트 및 삭제하도록 허용할 수도 있습니다.

이 정책은 `iotevents:ListInputs`, `iotevents:DescribeInput`, `iotevents:CreateInput`, `iotevents:DeleteInput` 및 `iotevents:UpdateInput` 권한을 사용자에게 부여합니다. 사용자에게 권한을 부여하고 콘솔을 사용하여 테스트하는 Amazon Simple Storage Service(Amazon S3)에 대한 예제 연습은 사용자 정책을 사용하여 버킷에 대한 액세스 제어를 참조하세요.

JSON

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Sid":"ListInputsInConsole",
         "Effect":"Allow",
         "Action":[
            "iotevents:ListInputs"
         ],
         "Resource":"arn:aws:iotevents:us-east-2:123456789012:input/*"
      },
      {
         "Sid":"ViewSpecificInputInfo",
```

```
            "Effect":"Allow",
            "Action":[
               "iotevents:DescribeInput"
            ],
            "Resource":"arn:aws:iotevents:us-east-1:123456789012:input/inputName"
         },
         {
            "Sid":"ManageInputs",
            "Effect":"Allow",
            "Action":[
               "iotevents:CreateInput",
               "iotevents:DeleteInput",
               "iotevents:DescribeInput",
               "iotevents:ListInputs",
               "iotevents:UpdateInput"
            ],
            "Resource":"arn:aws:iotevents:us-east-1:123456789012:input/*"
         }
      ]
}
```

태그를 기반으로 AWS IoT Events 입력 보기

태그는 AWS IoT Events 리소스를 구성하는 데 도움이 됩니다. 자격 증명 기반 정책의 조건을 사용하여 태그를 기반으로 AWS IoT Events 리소스에 대한 액세스를 제어할 수 있습니다. 이 예제에서는 ## 보기를 허용하는 정책을 생성할 수 있는 방법을 보여 줍니다. 하지만 ## 태그 Owner가 해당 사용자의 사용자 이름 값을 가지고 있는 경우에만 권한이 부여됩니다. 이 정책은 콘솔에서 이 작업을 완료하는 데 필요한 권한도 부여합니다.

JSON

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

```
            "Sid": "ViewInputsIfOwner",
            "Effect": "Allow",
            "Action": "iotevents:ListInputs",
            "Resource": "arn:aws:iotevents:*:*:input/*",
            "Condition": {
                "StringEquals": {"aws:ResourceTag/Owner": "${aws:username}"}
            }
        }
    ]
}
```

이 정책을 계정의 사용자에게 연결할 수 있습니다. 라는 사용자가 AWS IoT Events ### 보려고 `richard-roe` 하면 ##에 `Owner=richard-roe` 또는 태그를 지정해야 합니다`owner=richard-roe`. 그렇지 않으면 액세스가 거부됩니다. 조건 키 이름은 대소문자를 구분하지 않기 때문에 조건 태그 키 `Owner`는 `Owner` 및 `owner` 모두와 일치합니다. 자세한 내용은 IAM 사용자 설명서의 IAM JSON 정책 요소: 조건을 참조하세요.

에 대한 교차 서비스 혼동된 대리자 방지 AWS IoT Events

> **Note**
>
> - AWS IoT Events 서비스에서는 역할을 사용하여 리소스가 생성된 계정과 동일한 계정에서 작업을 시작할 수만 있습니다. 이렇게 하면 혼동된 대리자 공격을 방지하는 데 도움이 됩니다 AWS IoT Events.
> - 이 페이지는 혼동된 대리자 문제가 어떻게 작동하는지 확인할 수 있는 참조 역할을 하며 교차 계정 리소스가 서비스에서 허용된 경우 방지할 수 있습니다 AWS IoT Events .

혼동된 대리자 문제는 작업을 수행할 권한이 없는 엔터티가 권한이 더 많은 엔터티에게 작업을 수행하도록 강요할 수 있는 보안 문제입니다. 에서 AWS교차 서비스 가장은 혼동된 대리자 문제를 초래할 수 있습니다.

교차 서비스 가장은 한 서비스(직접 호출하는 서비스)가 다른 서비스(직접 호출되는 서비스)를 직접 호출할 때 발생할 수 있습니다. 직접 호출하는 서비스는 다른 고객의 리소스에 대해 액세스 권한이 없는 방식으로 작동하게 권한을 사용하도록 조작될 수 있습니다. 이를 방지하기 위해는 계정의 리소스에 대한 액세스 권한이 부여된 서비스 보안 주체를 사용하여 모든 서비스에 대한 데이터를 보호하는 데 도움이 되는 도구를 AWS 제공합니다.

리소스 정책에서 aws:SourceArn 및 aws:SourceAccount 전역 조건 컨텍스트 키를 사용하여 리소스에 다른 서비스를 AWS IoT Events 제공하는 권한을 제한하는 것이 좋습니다. 만약 aws:SourceArn 값에 Amazon S3 버킷 ARN과 같은 계정 ID가 포함되어 있지 않은 경우, 권한을 제한하려면 두 전역 조건 컨텍스트 키를 모두 사용해야 합니다. 두 전역 조건 컨텍스트 키와 계정을 포함한 aws:SourceArn 값을 모두 사용하는 경우, aws:SourceAccount 값 및 aws:SourceArn 값의 계정은 동일한 정책 명령문에서 사용할 경우 반드시 동일한 계정 ID를 사용해야 합니다.

하나의 리소스만 교차 서비스 액세스와 연결되도록 허용하려는 경우 aws:SourceArn을 사용하세요. 해당 계정의 모든 리소스가 교차 서비스 사용과 연결되도록 허용하려는 경우 aws:SourceAccount를 사용하십시오. aws:SourceArn의 값은 sts:AssumeRole 요청과 관련된 감지기 모델 또는 경보 모델이어야 합니다.

혼동된 대리인 문제로부터 보호하는 가장 효과적인 방법은 리소스의 전체 ARN이 포함된 aws:SourceArn글로벌 조건 컨텍스트 키를 사용하는 것입니다. 리소스의 전체 ARN을 모를 경우 또는 여러 리소스를 지정하는 경우, ARN의 알 수 없는 부분에 대해 와일드카드(*)를 포함한 aws:SourceArn전역 조건 컨텍스트 키를 사용합니다. 예제: arn:aws:*iotevents*:*:*123456789012*:*.

다음 예제에서는의 aws:SourceArn 및 aws:SourceAccount 전역 조건 컨텍스트 키를 사용하여 혼동된 대리자 문제를 AWS IoT Events 방지하는 방법을 보여줍니다.

주제

- 예: AWS IoT Events 감지기 모델에 대한 보안 액세스
- 예: AWS IoT Events 경보 모델에 대한 보안 액세스
- 예: 지정된 리전의 AWS IoT Events 리소스에 액세스
- 예:에 대한 로깅 옵션 구성 AWS IoT Events

예: AWS IoT Events 감지기 모델에 대한 보안 액세스

이 예제에서는의 특정 감지기 모델에 대한 액세스 권한을 안전하게 부여하는 IAM 정책을 생성하는 방법을 보여줍니다 AWS IoT Events. 이 정책은 조건을 사용하여 지정된 AWS 계정 및 AWS IoT Events 서비스만 역할을 수임하여 보안 계층을 추가할 수 있도록 합니다. 이 예에서 역할은 *WindTurbine01*이라는 감지기 모델에만 액세스할 수 있습니다.

JSON

```
{
```

```
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": [
                    "iotevents.amazonaws.com"
                ]
            },
            "Action": "sts:AssumeRole",
            "Condition": {
                "StringEquals": {
                "aws:SourceAccount": "123456789012"
                },
                "ArnEquals": {
                "aws:SourceArn": "arn:aws:iotevents:us-
east-1:123456789012:detectorModel/WindTurbine01"
                }
            }
        }
    ]
}
```

예: AWS IoT Events 경보 모델에 대한 보안 액세스

이 예제에서는가 경보 모델에 안전하게 액세스할 AWS IoT Events 수 있도록 허용하는 IAM 정책을 생성하는 방법을 보여줍니다. 이 정책은 조건을 사용하여 지정된 AWS 계정 및 AWS IoT Events 서비스만 역할을 수임할 수 있도록 합니다.

이 예제에서 역할은 경보 모델 ARN의 * 와일드카드로 표시된 대로 지정된 AWS 계정 내의 모든 경보 모델에 액세스할 수 있습니다. `aws:SourceAccount` 및 `aws:SourceArn` 조건은 혼동된 대리자 문제를 방지하기 위해 함께 작동합니다.

JSON

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

```
                "Service": [
                    "iotevents.amazonaws.com"
                ]
            },
            "Action": "sts:AssumeRole",
            "Condition": {
                "StringEquals": {
                "aws:SourceAccount": "123456789012"
                },
                "ArnEquals": {
                "aws:SourceArn": "arn:aws:iotevents:us-
east-1:123456789012:alarmModel/*"
                }
            }
        }
    ]
}
```

예: 지정된 리전의 AWS IoT Events 리소스에 액세스

이 예제에서는 특정 AWS 리전의 AWS IoT Events 리소스에 액세스하도록 IAM 역할을 구성하는 방법을 보여줍니다. IAM 정책에서 리전별 ARNs 사용하면 여러 지리적 영역의 AWS IoT Events 리소스에 대한 액세스를 제한할 수 있습니다. 이 접근 방식은 다중 리전 배포에서 보안 및 규정 준수를 유지하는 데 도움이 될 수 있습니다. 이 예제의 리전은 *us-east-1*입니다.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": [
                    "iotevents.amazonaws.com"
                ]
            },
            "Action": "sts:AssumeRole",
            "Condition": {
                "StringEquals": {
                "aws:SourceAccount": "123456789012"
```

```
                },
                "ArnEquals": {
                "aws:SourceArn": "arn:aws:iotevents:us-east-1:123456789012:*"
                }
            }
        }
    ]
}
```

예:에 대한 로깅 옵션 구성 AWS IoT Events

적절한 로깅은 애플리케이션을 모니터링, 디버깅 및 감사하는 데 AWS IoT Events 중요합니다. 이 섹션에서는에서 사용할 수 있는 로깅 옵션에 대한 개요를 제공합니다 AWS IoT Events.

이 예제에서는가 CloudWatch Logs AWS IoT Events 에 데이터를 로깅하도록 허용하는 IAM 역할을 구성하는 방법을 보여줍니다. 리소스 ARN에서 와일드카드(*)를 사용하면 AWS IoT Events 인프라 전체에서 포괄적인 로깅이 가능합니다.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": [
                    "iotevents.amazonaws.com"
                ]
            },
            "Action": "sts:AssumeRole",
            "Condition": {
                "StringEquals": {
                "aws:SourceAccount": "123456789012"
                },
                "ArnEquals": {
                "aws:SourceArn": "arn:aws:iotevents:us-east-1:123456789012:*"
                }
            }
        }
    ]
```

```
}
```

AWS IoT Events 자격 증명 및 액세스 문제 해결

다음 정보를 사용하여 및 IAM으로 작업할 때 발생할 수 있는 일반적인 문제를 진단 AWS IoT Events 하고 수정할 수 있습니다.

주제

- 에서 작업을 수행할 권한이 없음 AWS IoT Events
- iam:PassRole을 수행할 권한이 없음
- 내 외부의 사람이 내 AWS IoT Events 리소스에 액세스 AWS 계정 하도록 허용하고 싶습니다.

에서 작업을 수행할 권한이 없음 AWS IoT Events

에서 작업을 수행할 권한이 없다는 AWS Management Console 메시지가 표시되면 관리자에게 문의하여 도움을 받아야 합니다. 관리자는 사용자 이름과 암호를 제공한 사람입니다.

다음 예제 오류는 `mateojackson` IAM 사용자가 콘솔을 사용하여 *##*에 대한 세부 정보를 보려고 하지만 `iotevents:`*`ListInputs`* 권한이 없는 경우에 발생합니다.

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

이 경우, Mateo는 *`my-example-input`* 작업을 사용하여 `iotevents:`*`ListInput`* 리소스에 액세스하도록 허용하는 정책을 업데이트하라고 관리자에게 요청합니다.

`iam:PassRole`을 수행할 권한이 없음

`iam:PassRole` 작업을 수행할 수 있는 권한이 없다는 오류가 수신되면 AWS IoT Events에 역할을 전달할 수 있도록 정책을 업데이트해야 합니다.

일부 AWS 서비스 에서는 새 서비스 역할 또는 서비스 연결 역할을 생성하는 대신 기존 역할을 해당 서비스에 전달할 수 있습니다. 이렇게 하려면 역할을 서비스에 전달할 권한이 있어야 합니다.

다음 예 오류는 `marymajor`라는 IAM 사용자가 콘솔을 사용하여 AWS IoT Events에서 작업을 수행하려고 하는 경우에 발생합니다. 하지만 작업을 수행하려면 서비스 역할이 부여한 권한이 서비스에 있어야 합니다. Mary는 서비스에 역할을 전달할 권한이 없습니다.

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

이 경우, Mary가 `iam:PassRole`작업을 수행할 수 있도록 Mary의 정책을 업데이트해야 합니다.

도움이 필요한 경우 AWS 관리자에게 문의하세요. 관리자는 로그인 자격 증명을 제공한 사람입니다.

내 외부의 사람이 내 AWS IoT Events 리소스에 액세스 AWS 계정 하도록 허용하고 싶습니다.

다른 계정의 사용자 또는 조직 외부의 사람이 리소스에 액세스할 때 사용할 수 있는 역할을 생성할 수 있습니다. 역할을 수임할 신뢰할 수 있는 사람을 지정할 수 있습니다. 리소스 기반 정책 또는 액세스 제어 목록(ACL)을 지원하는 서비스의 경우, 이러한 정책을 사용하여 다른 사람에게 리소스에 대한 액세스 권한을 부여할 수 있습니다.

최상의 옵션을 결정하려면 다음 주제를 참조하세요.

- 에서 이러한 기능을 AWS IoT Events 지원하는지 여부를 알아보려면 섹션을 참조하세요AWS IoT Events 에서 IAM을 사용하는 방법.
- 소유 AWS 계정 한의 리소스에 대한 액세스 권한을 제공하는 방법을 알아보려면 IAM 사용 설명서의 소유한 다른의 IAM 사용자에게 액세스 권한 제공을 참조 AWS 계정 하세요.
- 리소스에 대한 액세스 권한을 타사에 제공하는 방법을 알아보려면 IAM 사용 설명서의 타사 AWS 계정 소유에 대한 액세스 권한 제공을 AWS 계정참조하세요.
- ID 페더레이션을 통해 액세스 권한을 제공하는 방법을 알아보려면 IAM 사용 설명서의 외부에서 인증된 사용자에게 액세스 권한 제공(ID 페더레이션)을 참조하세요.
- 크로스 계정 액세스에 대한 역할과 리소스 기반 정책 사용의 차이점을 알아보려면 IAM 사용 설명서의 IAM의 크로스 계정 리소스 액세스를 참조하세요.

안정성, 가용성 및 성능을 유지하기 AWS IoT Events 위한 모니터링

모니터링은 AWS IoT Events 및 AWS 솔루션의 안정성, 가용성 및 성능을 유지하는 데 중요한 부분입니다. 다중 지점 장애가 발생할 경우 보다 쉽게 디버깅할 수 있도록 AWS 솔루션의 모든 부분에서 모니터링 데이터를 수집해야 합니다. 모니터링을 시작하기 전에 다음 질문에 대한 답변을 포함하는 모니터링 계획을 생성해야 AWS IoT Events합니다.

- 모니터링의 목표

- 모니터링할 리소스
- 이러한 리소스를 모니터링하는 빈도
- 사용할 모니터링 도구
- 모니터링 작업을 수행할 사람
- 문제 발생 시 알려야 할 대상

다음 단계는 다양한 시간과 다양한 부하 조건에서 AWS IoT Events 성능을 측정하여 환경에서 정상적인 성능을 위한 기준을 설정하는 것입니다. AWS IoT Events가 과거 모니터링 데이터를 저장하는 것을 모니터링하면서 현재 성능 데이터를 이 과거 데이터와 비교하면 일반적인 성능 패턴과 성능 이상을 식별하고 이를 해결할 방법을 고안할 수 있습니다.

예를 들어 Amazon EC2를 사용하는 경우 인스턴스에 대해 CPU 사용률, 디스크 I/O 및 네트워크 사용률을 모니터링할 수 있습니다. 설정한 기준 이하로 성능이 떨어지면 인스턴스를 재구성하거나 최적화하여 CPU 사용률을 줄이거나 디스크 I/O를 개선하거나 네트워크 트래픽을 줄일 수 있습니다.

주제

- 모니터링할 수 있는 도구 AWS IoT Events
- Amazon CloudWatch AWS IoT Events 를 사용한 모니터링
- 를 사용하여 AWS IoT Events API 호출 로깅 AWS CloudTrail

모니터링할 수 있는 도구 AWS IoT Events

AWS 는 모니터링에 사용할 수 있는 다양한 도구를 제공합니다 AWS IoT Events. 이들 도구 중에는 모니터링을 자동으로 수행하도록 구성할 수 있는 도구도 있지만, 수동 작업이 필요한 도구도 있습니다. 모니터링 작업은 최대한 자동화하는 것이 좋습니다.

자동 모니터링 도구

다음과 같은 자동 모니터링 도구를 사용하여 문제 발생 시 모니터링하고 AWS IoT Events 보고할 수 있습니다.

- Amazon CloudWatch Logs - AWS CloudTrail 또는 기타 소스에서 로그 파일을 모니터링, 저장 및 액세스합니다. 자세한 내용은 Amazon CloudWatch 사용 설명서의 Amazon CloudWatch 대시보드 사용을 참조하세요.
- AWS CloudTrail 로그 모니터링 - 계정 간에 로그 파일을 공유하고, CloudTrail 로그 파일을 CloudWatch Logs로 전송하여 실시간으로 모니터링하고, Java로 로그 처리 애플리케이션을 작성

하고, CloudTrail에서 전송한 후 로그 파일이 변경되지 않았는지 확인합니다. 자세한 내용은 AWS CloudTrail 사용 설명서의 CloudTrail 로그 파일 작업을 참조하십시오.

수동 모니터링 도구

모니터링의 또 다른 중요한 부분은 CloudWatch 경보 AWS IoT Events 가 다루지 않는 항목을 수동으로 모니터링하는 것입니다. AWS IoT Events, CloudWatch 및 기타 AWS 콘솔 대시보드는 AWS 환경 상태를 at-a-glance 볼 수 있습니다. 로그 파일도 확인하는 것이 좋습니다 AWS IoT Events.

- AWS IoT Events 콘솔에 다음이 표시됩니다.
  - 감지기 모델 수
  - 탐지기
  - 입력
  - Settings
- CloudWatch 홈 페이지에는 다음 내용이 표시됩니다.
  - 현재 경보 및 상태
  - 경보 및 리소스 그래프
  - 서비스 상태

  또한 CloudWatch를 사용하여 다음을 수행할 수 있습니다.
  - 생성 CloudWatch 대시보드를 생성하여 관심 있는 서비스 모니터링
  - 지표 데이터를 그래프로 작성하여 문제를 해결하고 추세 파악
  - 모든 AWS 리소스 지표 검색 및 찾아보기
  - 문제에 대해 알려주는 경보 생성 및 편집

Amazon CloudWatch AWS IoT Events 를 사용한 모니터링

AWS IoT Events 감지기 모델을 개발하거나 디버깅할 때 어떤 작업이 AWS IoT Events 수행되고 있는지, 그리고 발생하는 모든 오류를 알아야 합니다. Amazon CloudWatch는 AWS 리소스와 AWS 에서 실행하는 애플리케이션을 실시간으로 모니터링합니다. CloudWatch를 사용하면 시스템 전체의 리소스 사용률, 애플리케이션 성능, 운영 상태를 파악할 수 있습니다. AWS IoT Events 감지기 모델 개발 시 Amazon CloudWatch 로깅 활성화에는 AWS IoT Events의 CloudWatch 로깅을 활성화하는 방법에 대한 정보가 있습니다. 아래 표시된 것과 같은 로그를 생성하려면 상세 수준을 '디버그'로 설정하고 탐지기 모델 이름과 선택적 KeyValue인 디버그 대상을 하나 이상 제공해야 합니다.

다음은 AWS IoT Events에서 생성한 CloudWatch 디버그 수준 로그 항목의 예입니다.


```
{
  "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
    }
  ]
}
```

를 사용하여 AWS IoT Events API 호출 로깅 AWS CloudTrail

AWS IoT Events 는 사용자 AWS CloudTrail, 역할 또는 서비스가 수행한 작업에 대한 레코드를 제공하는 AWS 서비스와 통합됩니다 AWS IoT Events. CloudTrail은 AWS IoT Events 콘솔의 호출 및 API에 대한 코드 호출을 포함하여에 대한 모든 API 호출을 이벤트 AWS IoT Events 로 캡처합니다. AWS IoT Events APIs

추적을 생성하면 이벤트를 포함하여 CloudTrail 이벤트를 지속적으로 Amazon S3 버킷에 배포할 수 있습니다 AWS IoT Events. 추적을 구성하지 않은 경우에도 이벤트 기록에서 CloudTrail 콘솔의 최신 이벤트를 볼 수 있습니다. CloudTrail에서 수집한 정보를 사용하여 수행된 요청, 요청이 수행된 AWS IoT Events IP 주소, 요청을 수행한 사람, 요청이 수행된 시간 및 추가 세부 정보를 확인할 수 있습니다.

CloudTrail에 대한 자세한 설명은 AWS CloudTrail 사용자 가이드를 참조하십시오.

AWS IoT Events CloudTrail의 정보

AWS 계정을 생성할 때 계정에서 CloudTrail이 활성화됩니다. 활동이 발생하면 AWS IoT Events해당 활동이 이벤트 기록에 다른 AWS 서비스 이벤트와 함께 CloudTrail 이벤트에 기록됩니다. AWS 계정에서 최근 이벤트를 보고 검색하고 다운로드할 수 있습니다. 자세한 설명은 CloudTrail 이벤트 기록 작업을 참조하세요.

에 대한 이벤트를 포함하여 AWS 계정에 이벤트를 지속적으로 기록하려면 추적을 AWS IoT Events생성합니다. CloudTrail은 추적을 사용하여 Amazon S3 버킷으로 로그 파일을 전송할 수 있습니다. 기본적으로 콘솔에서 추적을 생성하면 추적이 모든 AWS 리전에 적용됩니다. 추적은 AWS 파티션의 모든 리전에서 이벤트를 로깅하고 지정한 Amazon S3 버킷으로 로그 파일을 전송합니다. 또한 CloudTrail 로그에서 수집된 이벤트 데이터를 추가로 분석하고 처리하도록 다른 AWS 서비스를 구성할 수 있습니다. 자세한 내용은 다음을 참조하세요.

- AWS 계정에 대한 추적 생성
- CloudTrail 지원 서비스 및 통합
- CloudTrail에 대한 Amazon SNS 알림 구성
- 여러 리전에서 CloudTrail 로그 파일 받기 및 여러 계정에서 CloudTrail 로그 파일 받기

모든 이벤트 또는 로그 항목에는 요청을 생성했던 사용자에 관한 정보가 포함됩니다. ID 정보를 이용하면 다음을 쉽게 판단할 수 있습니다.

- 요청을 루트로 했는지 아니면 IAM 사용자 자격 증명 정보로 했는지 여부.
- 역할 또는 페더레이션 사용자의 임시 자격 증명을 사용하여 요청이 생성되었는지 여부.
- 요청이 다른 AWS 서비스에 의해 이루어졌는지 여부입니다.

자세한 내용은 API 참조에 설명된 CloudTrail userIdentity element. AWS IoT Events actions를 AWS IoT Events 참조하세요.

AWS IoT Events 로그 파일 항목 이해

추적은 사용자가 지정한 Amazon S3 버킷에 로그 파일로 이벤트를 전송할 수 있도록 하는 구성입니다. AWS CloudTrail 로그 파일에는 하나 이상의 로그 항목이 포함됩니다. 이벤트는 모든 소스로부터의 단일 요청을 나타내며 요청 작업, 작업 날짜와 시간, 요청 파라미터 등에 대한 정보가 들어 있습니다. CloudTrail 로그 파일은 퍼블릭 API 호출에 대한 순서 지정된 스택 추적이 아니기 때문에 특정 순서로 표시되지 않습니다.

AWS 계정에서 CloudTrail 로깅을 활성화하면 AWS IoT Events 작업에 대한 대부분의 API 호출이 다른 AWS 서비스 레코드와 함께 기록되는 CloudTrail 로그 파일에서 추적됩니다. CloudTrail은 기간 및 파일 크기를 기준으로 새 파일을 만들고 기록하는 시점을 결정합니다.

모든 로그 항목은 누가 요청을 생성했는가에 대한 정보가 들어 있습니다. 로그 항목의 사용자 신원 정보를 이용하면 다음을 쉽게 판단할 수 있습니다.

- 요청을 루트로 했는지 아니면 IAM 사용자 자격 증명 정보로 했는지 여부.
- 역할 또는 페더레이션 사용자의 임시 자격 증명을 사용하여 요청이 생성되었는지 여부.
- 요청이 다른 AWS 서비스에 의해 이루어졌는지 여부입니다.

원하는 기간만큼 Amazon S3 버킷에 로그 파일을 저장할 수 있습니다. 그러나 Amazon S3 수명 주기 규칙을 정의하여 자동으로 로그 파일을 보관하거나 삭제할 수도 있습니다. 기본적으로 로그 파일은 Amazon S3 서버 측 암호화(SSE)를 통해 암호화합니다.

새 로그 파일이 전달되면 Amazon SNS 알림이 게시되도록 CloudTrail을 구성하여 로그 파일 전송 시 알림을 받을 수 있습니다. 자세한 내용은 CloudTrail용 Amazon SNS 알림 구성을 참조하십시오.

여러 AWS 리전 및 AWS 여러 계정의 AWS IoT Events 로그 파일을 단일 Amazon S3 버킷으로 집계할 수도 있습니다.

자세한 내용은 여러 리전에서 CloudTrail 로그 파일 받기와 여러 계정에서 CloudTrail 로그 파일 받기를 참조하십시오.

예: CloudTrail에 대한 DescribeDetector 작업

다음은 `DescribeDetector` 작업을 설명하는 CloudTrail 로그 항목을 보여 주는 예시입니다.


```
{
  "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"
```

```
}
```

예: CloudTrail에 대한 CreateDetectorModel 작업

다음은 `CreateDetectorModel` 작업을 설명하는 CloudTrail 로그 항목을 보여 주는 예시입니다.

```
{
  "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"
}
```

예: CloudTrail에 대한 CreateInput 작업

다음은 `CreateInput` 작업을 설명하는 CloudTrail 로그 항목을 보여 주는 예시입니다.

```
{
  "eventVersion": "1.05",
  "userIdentity": {
    "type": "AssumedRole",
    "principalId": "AKIAI44QH8DHBEXAMPLE:IotEvents-Lambda",
    "arn": "arn:aws:sts::123456789012:assumed-role/IotEventsLambda-RoleForIotEvents-
ABC123DEF456/IotEvents-Lambda",
    "accountId": "123456789012",
    "accessKeyId": "AKIAI44QH8DHBEXAMPLE",
    "sessionContext": {
      "attributes": {
        "mfaAuthenticated": "false",
        "creationDate": "2019-02-07T22:22:30Z"
      },
      "sessionIssuer": {
        "type": "Role",
        "principalId": "AKIAI44QH8DHBEXAMPLE",
        "arn": "arn:aws:iam::123456789012:role/IotEventsLambda-RoleForIotEvents-
ABC123DEF456",
        "accountId": "123456789012",
        "userName": "IotEventsLambda-RoleForIotEvents-ABC123DEF456"
      }
    }
  },
  "eventTime": "2019-02-07T23:54:43Z",
  "eventSource": "iotevents.amazonaws.com",
  "eventName": "CreateInput",
  "awsRegion": "us-east-1",
  "sourceIPAddress": "192.168.0.1",
  "userAgent": "aws-internal/3",
  "requestParameters": {
    "inputName": "batchputmessagedetectorupdated",
    "inputDescription": "batchputmessagedetectorupdated"
  },
  "responseElements": null,
  "requestID": "fb315af4-39e9-4114-94d1-89c9183394c1",
```

```
  "eventID": "6d8cf67b-2a03-46e6-bbff-e113a7bded1e",
  "eventType": "AwsApiCall",
  "recipientAccountId": "123456789012"
}
```

예: CloudTrail에 대한 DeleteDetectorModel 작업

다음은 `DeleteDetectorModel` 작업을 설명하는 CloudTrail 로그 항목을 보여 주는 예시입니다.

```
{
  "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"
}
```

예: CloudTrail에 대한 DeleteInput 작업

다음은 `DeleteInput` 작업을 설명하는 CloudTrail 로그 항목을 보여 주는 예시입니다.

```
{
  "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"
}
```

예: CloudTrail에 대한 DescribeDetectorModel 작업

다음은 `DescribeDetectorModel` 작업을 설명하는 CloudTrail 로그 항목을 보여 주는 예시입니다.

```
{
  "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"
}
```

예: CloudTrail에 대한 DescribeInput 작업

다음은 `DescribeInput` 작업을 설명하는 CloudTrail 로그 항목을 보여 주는 예시입니다.

```
{
  "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"
}
```

예: CloudTrail에 대한 DescribeLoggingOptions 작업

다음은 `DescribeLoggingOptions` 작업을 설명하는 CloudTrail 로그 항목을 보여 주는 예시입니다.

```
{
  "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"
}
```

예: CloudTrail에 대한 ListDetectorModels 작업

다음은 `ListDetectorModels` 작업을 설명하는 CloudTrail 로그 항목을 보여 주는 예시입니다.

```
{
  "eventVersion": "1.05",
  "userIdentity": {
    "type": "AssumedRole",
    "principalId": "AKIAI44QH8DHBEXAMPLE:IotEvents-EventsLambda",
    "arn": "arn:aws:sts::123456789012:assumed-role/IotEventsLambda-RoleForIotEvents-
ABCD123DEF456/IotEvents-EventsLambda",
    "accountId": "123456789012",
    "accessKeyId": "AKIAI44QH8DHBEXAMPLE",
    "sessionContext": {
      "attributes": {
        "mfaAuthenticated": "false",
        "creationDate": "2019-02-07T22:22:30Z"
      },
      "sessionIssuer": {
        "type": "Role",
        "principalId": "AKIAI44QH8DHBEXAMPLE",
        "arn": "arn:aws:iam::123456789012:role/IotEventsLambda-RoleForIotEvents-
ABCD123DEF456",
        "accountId": "123456789012",
        "userName": "IotEventsLambda-RoleForIotEvents-ABCD123DEF456"
      }
    }
  },
  "eventTime": "2019-02-07T23:53:23Z",
  "eventSource": "iotevents.amazonaws.com",
  "eventName": "ListDetectorModels",
  "awsRegion": "us-east-1",
  "sourceIPAddress": "192.168.0.1",
  "userAgent": "aws-internal/3",
  "requestParameters": {
    "nextToken": "CkZEZXRlY3Rvck1vZGVsMl9saXN0ZGV0ZWN0b3Jtb2RlbHN0ZXN0X2VlOWJkZTk1YT",
    "maxResults": 3
  },
```

```
  "responseElements": null,
  "requestID": "6d70f262-da95-4bb5-94b4-c08369df75bb",
  "eventID": "2d01a25c-d5c7-4233-99fe-ce1b8ec05516",
  "eventType": "AwsApiCall",
  "recipientAccountId": "123456789012"
}
```

예: CloudTrail에 대한 ListDetectorModelVersions 작업

다음은 `ListDetectorModelVersions` 작업을 설명하는 CloudTrail 로그 항목을 보여 주는 예시입니다.

```
{
  "eventVersion": "1.05",
  "userIdentity": {
    "type": "AssumedRole",
    "principalId": "AKIAI44QH8DHBEXAMPLE:IotEvents-EventsLambda",
    "arn": "arn:aws:sts::123456789012:assumed-role/IotEventsLambda-RoleForIotEvents-
ABCD123DEF456/IotEvents-EventsLambda",
    "accountId": "123456789012",
    "accessKeyId": "AKIAI44QH8DHBEXAMPLE",
    "sessionContext": {
      "attributes": {
        "mfaAuthenticated": "false",
        "creationDate": "2019-02-07T22:22:30Z"
      },
      "sessionIssuer": {
        "type": "Role",
        "principalId": "AKIAI44QH8DHBEXAMPLE",
        "arn": "arn:aws:iam::123456789012:role/IotEventsLambda-RoleForIotEvents-
ABCD123DEF456",
        "accountId": "123456789012",
        "userName": "IotEventsLambda-RoleForIotEvents-ABCD123DEF456"
      }
    }
  },
  "eventTime": "2019-02-07T23:53:33Z",
  "eventSource": "iotevents.amazonaws.com",
  "eventName": "ListDetectorModelVersions",
  "awsRegion": "us-east-1",
  "sourceIPAddress": "192.168.0.1",
  "userAgent": "aws-internal/3",
  "requestParameters": {
```

```
    "detectorModelName": "myDetectorModel",
    "maxResults": 2
  },
  "responseElements": null,
  "requestID": "ebecb277-6bd8-44ea-8abd-fbf40ac044ee",
  "eventID": "fc6281a2-3fac-4e1e-98e0-ca6560b8b8be",
  "eventType": "AwsApiCall",
  "recipientAccountId": "123456789012"
}
```

예: CloudTrail에 대한 ListDetectors 작업

다음은 `ListDetectors` 작업을 설명하는 CloudTrail 로그 항목을 보여 주는 예시입니다.

```
{
  "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"
}
```

예: CloudTrail에 대한 ListInputs 작업

다음은 `ListInputs` 작업을 설명하는 CloudTrail 로그 항목을 보여 주는 예시입니다.

```
{
  "eventVersion": "1.05",
  "userIdentity": {
    "type": "AssumedRole",
    "principalId": "AKIAI44QH8DHBEXAMPLE:IotEvents-EventsLambda",
    "arn": "arn:aws:sts::123456789012:assumed-role/IotEventsLambda-RoleForIotEvents-
ABCD123DEF456/IotEvents-EventsLambda",
    "accountId": "123456789012",
    "accessKeyId": "AKIAI44QH8DHBEXAMPLE",
    "sessionContext": {
      "attributes": {
        "mfaAuthenticated": "false",
        "creationDate": "2019-02-07T22:22:30Z"
      },
      "sessionIssuer": {
        "type": "Role",
        "principalId": "AKIAI44QH8DHBEXAMPLE",
        "arn": "arn:aws:iam::123456789012:role/IotEventsLambda-RoleForIotEvents-
ABCD123DEF456",
        "accountId": "123456789012",
        "userName": "IotEventsLambda-RoleForIotEvents-ABCD123DEF456"
      }
    }
  },
  "eventTime": "2019-02-07T23:53:57Z",
  "eventSource": "iotevents.amazonaws.com",
  "eventName": "ListInputs",
  "awsRegion": "us-east-1",
  "sourceIPAddress": "192.168.0.1",
```

```
  "userAgent": "aws-internal/3",
  "requestParameters": {
    "nextToken": "CkhjYW5hcnlfdGVzdF9pbnB1dF9saXN0ZGV0ZWN0b3Jtb2RlbHN0ZXN0ZDU3OGZ",
    "maxResults": 3
  },
  "responseElements": null,
  "requestID": "dd6762a1-1f24-4e63-a986-5ea3938a03da",
  "eventID": "c500f6d8-e271-4366-8f20-da4413752469",
  "eventType": "AwsApiCall",
  "recipientAccountId": "123456789012"
}
```


예: CloudTrail에 대한 PutLoggingOptions 작업

다음은 PutLoggingOptions 작업을 설명하는 CloudTrail 로그 항목을 보여 주는 예시입니다.


```
{
  "eventVersion": "1.05",
  "userIdentity": {
    "type": "AssumedRole",
    "principalId": "AKIAI44QH8DHBEXAMPLE:IotEvents-EventsLambda",
    "arn": "arn:aws:sts::123456789012:assumed-role/IotEventsLambda-RoleForIotEvents-
ABCD123DEF456/IotEvents-EventsLambda",
    "accountId": "123456789012",
    "accessKeyId": "AKIAI44QH8DHBEXAMPLE",
    "sessionContext": {
      "attributes": {
        "mfaAuthenticated": "false",
        "creationDate": "2019-02-07T22:22:30Z"
      },
      "sessionIssuer": {
        "type": "Role",
        "principalId": "AKIAI44QH8DHBEXAMPLE",
        "arn": "arn:aws:iam::123456789012:role/IotEventsLambda-RoleForIotEvents-
ABCD123DEF456",
        "accountId": "123456789012",
        "userName": "IotEventsLambda-RoleForIotEvents-ABCD123DEF456"
      }
    }
  },
  "eventTime": "2019-02-07T23:56:43Z",
  "eventSource": "iotevents.amazonaws.com",
  "eventName": "PutLoggingOptions",
  "awsRegion": "us-east-1",
```

```
  "sourceIPAddress": "192.168.0.1",
  "userAgent": "aws-internal/3",
  "requestParameters": {
    "loggingOptions": {
      "roleArn": "arn:aws:iam::123456789012:role/logging__logging_role",
      "level": "INFO",
      "enabled": false
    }
  },
  "responseElements": null,
  "requestID": "df570e50-fb19-4636-9ec0-e150a94bc52c",
  "eventID": "3247f928-26aa-471e-b669-e4a9e6fbc42c",
  "eventType": "AwsApiCall",
  "recipientAccountId": "123456789012"
}
```

예: CloudTrail에 대한 UpdateDetectorModel 작업

다음은 `UpdateDetectorModel` 작업을 설명하는 CloudTrail 로그 항목을 보여 주는 예시입니다.

```
{
  "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"
}
```

예: CloudTrail에 대한 UpdateInput 작업

다음은 UpdateInput 작업을 설명하는 CloudTrail 로그 항목을 보여 주는 예시입니다.

```
{
  "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"
}
```

예: CloudTrail에 대한 BatchPutMessage 작업

AWS IoT Events 는 데이터 영역 API 로깅에 CloudTrail 통합을 사용할 수 있습니다. 이 예제에서는 `BatchPutMessage` 작업을 통해 데이터 이벤트에 대한 세부 정보를 추가합니다.

```
{
            "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"
        }
```

```
        },
```

에 대한 규정 준수 검증 AWS IoT Events

AWS 서비스 가 특정 규정 준수 프로그램의 범위 내에 있는지 알아보려면 AWS 서비스 규정 준수 프로그램 범위 내를 참조하고 관심 있는 규정 준수 프로그램을 선택합니다. 일반 정보는 AWS 규정 준수 프로그램.

를 사용하여 타사 감사 보고서를 다운로드할 수 있습니다 AWS Artifact. 자세한 내용은 에서 보고서 다운로드 AWS Artifact에서 .

사용 시 규정 준수 책임은 데이터의 민감도, 회사의 규정 준수 목표, 관련 법률 및 규정에 따라 AWS 서비스 결정됩니다. 사용 시 규정 준수 책임에 대한 자세한 내용은 AWS 보안 설명서를 AWS 서비스참조하세요.

의 복원력 AWS IoT Events

AWS 글로벌 인프라는 AWS 리전 및 가용 영역을 기반으로 구축됩니다. AWS 리전은 물리적으로 분리되고 격리된 다수의 가용 리전을 제공하며 이러한 가용 리전은 짧은 지연 시간, 높은 처리량 및 높은 중복성을 갖춘 네트워크에 연결되어 있습니다. 가용 영역을 사용하면 중단 없이 가용 영역 간에 자동으로 장애 조치가 이루어지는 애플리케이션 및 데이터베이스를 설계하고 운영할 수 있습니다. 가용 영역은 기존의 단일 또는 복수 데이터 센터 인프라보다 가용성, 내결함성, 확장성이 뛰어납니다.

AWS 리전 및 가용 영역에 대한 자세한 내용은 AWS 글로벌 인프라를 참조하세요.

의 인프라 보안 AWS IoT Events

관리형 서비스인는 AWS 글로벌 네트워크 보안으로 보호 AWS IoT Events 됩니다. AWS 보안 서비스 및가 인프라를 AWS 보호하는 방법에 대한 자세한 내용은 AWS 클라우드 보안을 참조하세요. 인프라 보안 모범 사례를 사용하여 환경을 설계하려면 보안 원칙 AWS Well-Architected Framework의 인프라 보호를 참조하세요 AWS .

AWS 에서 게시한 API 호출을 사용하여 네트워크를 AWS IoT Events 통해에 액세스합니다. 고객은 다음을 지원해야 합니다.

- Transport Layer Security(TLS) TLS 1.2는 필수이며 TLS 1.3을 권장합니다.
- DHE(Ephemeral Diffie-Hellman) 또는 ECDHE(Elliptic Curve Ephemeral Diffie-Hellman)와 같은 완전 전송 보안(PFS)이 포함된 암호 제품군 Java 7 이상의 최신 시스템은 대부분 이러한 모드를 지원합니다.

AWS AWS IoT Events 리소스에 대한 서비스 할당량

이 AWS 일반 참조 안내서에서는 AWS IoT Events AWS 계정의 기본 할당량을 제공합니다. 지정하지 않는 한 각 할당량은 AWS 리전당입니다. 자세한 내용은 AWS 일반 참조 안내서에서 AWS IoT Events 엔드포인트 및 할당량 및 AWS Service Quotas를 참조하십시오.

서비스 할당량 증가를 요청하려면 지원 센터 콘솔에서 지원 사례를 제출하십시오. 자세한 내용은 Service Quotas User Guide(Service Quotas 사용 설명서)의 Requesting a quota increase(할당량 증가 요청)를 참조하십시오.

> **Note**
>
> - 계정 내 감지기 모델 및 입력의 모든 이름은 고유해야 합니다.
> - 감지기 모델 및 입력이 생성된 후에는 이름을 변경할 수 없습니다.

AWS IoT Events 리소스에 태그 지정

감지기 모델 및 입력을 쉽게 관리 및 구성하기 위해 이러한 각 리소스에 고유한 메타데이터를 태그의 형태로 할당할 수 있습니다. 이 단원에서는 태그를 설명하고 태그를 생성하는 방법을 보여 줍니다.

태그 기본 사항

태그를 사용하면 용도, 소유자 또는 환경 등 다양한 방식으로 AWS IoT Events 리소스를 분류할 수 있습니다. 이는 동일한 유형의 리소스가 많을 때 유용합니다. 지정한 태그를 기반으로 특정 리소스를 신속하게 식별할 수 있습니다.

각 태그는 사용자가 정의하는 키와 선택적 값으로 구성됩니다. 예를 들어, 입력에 대한 태그 세트를 정의하여 이러한 입력을 전송하는 디바이스를 유형별로 추적할 수 있습니다. 각 리소스 유형에 대한 요건을 충족하는 태그 키 세트를 생성하는 것이 좋습니다. 일관된 태그 키 세트를 사용하면 리소스를 보다 쉽게 관리할 수 있습니다.

AWS IoT 개발자 안내서의 IAM 정책에서 태그 사용에 설명된 대로 추가하거나 적용한 태그를 기반으로 리소스를 검색 및 필터링하고, 태그를 사용하여 비용을 분류 및 추적하고, 태그를 사용하여 리소스에 대한 액세스를 제어할 수 있습니다.

사용 편의성을 위해의 Tag Editor는 태그를 생성하고 관리하는 중앙 통합 방법을 AWS Management Console 제공합니다. 자세한 내용은 AWS 리소스 태그 지정 및 태그 편집기 사용 설명서의 태그 편집기 시작하기를 참조하세요.

AWS CLI 및 AWS IoT Events API를 사용하여 태그로 작업할 수도 있습니다. 다음 명령에서 "Tags" 필드를 사용하여 태그를 만들 때 태그를 감지기 모델 및 입력과 연결할 수 있습니다.

- 감지기 모델 생성
- 입력 생성

다음 명령을 사용하여 태깅을 지원하는 기존 리소스에 대해 태그를 추가, 수정, 삭제할 수 있습니다.

- TagResource
- ListTagsForResource
- UntagResource

태그 키와 값을 편집할 수 있으며 언제든지 리소스에서 태그를 제거할 수 있습니다. 태그의 값을 빈 문자열로 설정할 수 있지만 태그의 값을 Null로 설정할 수는 없습니다. 해당 리소스의 기존 태그와 동일한 키를 가진 태그를 추가하면 새 값이 이전 값을 덮어씁니다. 리소스를 삭제하면, 리소스에 대한 연결이 완료된 태그 또한 삭제됩니다.

자세한 내용은 AWS 리소스 태그 지정 모범 사례를 참조하세요.

태그 규제 및 제한

태그에 적용되는 기본 제한 사항은 다음과 같습니다.

- 리소스당 최대 태그 수 - 50
- 최대 키 길이 - UTF-8 형식의 유니코드 문자 127자
- 최대 값 길이 - UTF-8 형식의 유니코드 문자 255자
- 태그 키와 값은 대소문자를 구분합니다.
- 태그 이름 또는 값에 "aws:" 접두사는 AWS 사용하도록 예약되어 있으므로 사용하지 마십시오. 이 접두사가 지정된 태그 이름이나 값은 편집하거나 삭제할 수 없습니다. 이 접두사가 지정된 태그는 리소스당 태그 수 제한에 포함되지 않습니다.
- 태그 지정 스키마를 여러 서비스와 리소스에서 사용하는 경우 다른 서비스에서 허용되는 문자에 제한이 있을 수 있음에 유의하세요. 일반적으로 허용되는 문자는 UTF-8로 표현할 수 있는 문자, 공백 및 숫자와 + - = . _ : / @ 등의 특수 문자입니다.

IAM 정책에 태그 사용

AWS IoT Events API 작업에 사용하는 IAM 정책에 태그 기반의 리소스 수준 권한을 적용할 수 있습니다. 이를 통해 사용자가 생성, 수정 또는 사용할 수 있는 리소스를 더욱 정확하게 제어할 수 있습니다.

리소스 태그를 기반으로 사용자 액세스(권한)를 제어하기 위해 IAM 정책에서 다음 조건 콘텍스트 키 및 값과 함께 `Condition` 요소(`Condition` 블록)를 사용합니다.

- `aws:ResourceTag/`*`<tag-key>`*`: `*`<tag-value>`*를 사용하여 특정 태그가 지정된 리소스에 대한 사용자 작업을 허용 또는 거부합니다.
- `aws:RequestTag/`*`<tag-key>`*`: `*`<tag-value>`*를 사용하여 태그를 허용하는 리소스를 생성하거나 수정하는 API 요청을 작성할 때 특정 태그를 사용하도록(또는 사용하지 않도록) 요구합니다.
- `aws:TagKeys: [`*`<tag-key>`*`, ...]`를 사용하여 태깅 가능한 리소스를 생성하거나 수정하는 API 요청을 작성할 때 특정 태그 키 집합을 사용하도록(또는 사용하지 않도록) 요구합니다.

> **Note**
>
> IAM 정책의 조건 컨텍스트 키와 값은 태깅 가능한 리소스의 ID가 필수 파라미터인 AWS IoT Events 작업에만 적용됩니다.

태그 사용에 대한 자세한 내용은 AWS Identity and Access Management 사용 설명서의 태그를 사용한 액세스 제어를 참조하십시오. 이 설명서의 IAM JSON 정책 참조 단원에서는 IAM에서 JSON 정책의 자세한 구문과 설명, 요소의 예, 변수, 평가 로직을 설명합니다.

다음은 태그 기반 제한 2개를 적용하는 정책 예제입니다. 이 정책으로 제한되는 사용자는 다음과 같습니다.

- 리소스에 태그 "env=prod"를 지정할 수 없습니다. 이 예제에서는 `"aws:RequestTag/env" : "prod"` 행을 참조하십시오.
- 기존 태그 "env=prod"가 지정된 리소스를 수정 또는 액세스할 수 없습니다. 이 예제에서는 `"aws:ResourceTag/env" : "prod"` 행을 참조하십시오.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Deny",
            "Action": [
                "iotevents:CreateDetectorModel",
                "iotevents:CreateAlarmModel",
                "iotevents:CreateInput",
                "iotevents:TagResource"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "aws:RequestTag/env": "prod"
                }
            }
        },
        {
            "Effect": "Deny",
```

```
            "Action": [
                "iotevents:DescribeDetectorModel",
                "iotevents:DescribeAlarmModel",
                "iotevents:UpdateDetectorModel",
                "iotevents:UpdateAlarmModel",
                "iotevents:DeleteDetectorModel",
                "iotevents:DeleteAlarmModel",
                "iotevents:ListDetectorModelVersions",
                "iotevents:ListAlarmModelVersions",
                "iotevents:UpdateInput",
                "iotevents:DescribeInput",
                "iotevents:DeleteInput",
                "iotevents:ListTagsForResource",
                "iotevents:TagResource",
                "iotevents:UntagResource",
                "iotevents:UpdateInputRouting"
            ],
            "Resource": "*",
            "Condition": {
                "StringLike": {
                    "aws:ResourceTag/env": "prod"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "iotevents:*"
            ],
            "Resource": "*"
        }
    ]
}
```

또한 다음과 같이 목록에서 태그를 둘러싸 지정된 태그 키에 대해 여러 태그 값을 지정할 수도 있습니다.

```
        "StringEquals" : {
          "aws:ResourceTag/env" : ["dev", "test"]
        }
```

> **Note**
>
> 태그를 기준으로 리소스에 대한 사용자 액세스를 허용 또는 거부하는 경우 동일한 리소스에서 태그를 추가 또는 제거할 수 있도록 사용자를 명시적으로 거부할 것을 고려해야 합니다. 그렇지 않으면 사용자가 제한을 피해 태그를 수정하여 리소스에 대한 액세스 권한을 얻을 수 있습니다.

문제 해결 AWS IoT Events

이 문제 해결 안내서는를 사용할 때 발생할 수 있는 일반적인 문제에 대한 솔루션을 제공합니다 AWS IoT Events. 주제를 탐색하여 이벤트 감지, 데이터 액세스, 권한, 서비스 통합, 디바이스 구성 등의 문제를 식별하고 해결합니다. 이 안내서는 AWS IoT Events 콘솔, API, CLI, 오류, 지연 시간 및 통합에 대한 문제 해결 조언을 통해 문제를 신속하게 해결하여 안정적이고 확장 가능한 이벤트 기반 애플리케이션을 구축하는 것을 목표로 합니다.

주제

일반적인 AWS IoT Events 문제 및 해결 방법

오류를 해결하고 문제를 해결할 수 있는 해결 방법을 찾으려면 다음 섹션을 참조하세요 AWS IoT Events.

오류

감지기 모델 생성 오류

감지기 모델을 생성하려고 하면 오류가 발생합니다.

Solution

감지기 모델을 생성할 때 다음 제한 사항을 고려해야 합니다.

- 각 `action` 필드에는 한 가지 작업만 허용됩니다.
- `transitionEvents`에서 `condition`은(는) 필수입니다. `OnEnter`, `OnInput` 및 `OnExit` 이벤트는 선택 사항입니다.
- `condition` 필드가 비어 있는 경우 조건 표현식의 평가 결과는 `true`와 같습니다.
- 조건 표현식을 평가 결과는 부울 값이어야 합니다. 결과가 부울 값이 아닌 경우 이 값은 `false`와 같으며 `actions` 또는 이벤트에서 지정된 `nextState`로 전환을 트리거하지 않습니다.

자세한 내용은 AWS IoT Events 감지기 모델 제한 사항 및 제한 사항 섹션을 참조하십시오.

삭제된 감지기 모델 업데이트

몇 분 전에 감지기 모델을 업데이트하거나 삭제했지만 MQTT 메시지 또는 SNS 경보를 통해 이전 감지기 모델의 상태 업데이트를 계속 받고 있습니다.

Solution

감지기 모델을 업데이트, 삭제 또는 재생성하는 경우 (UpdateDetectorModel 참조) 모든 감지기 인스턴스가 삭제되고 새 모델이 사용되기까지 지연이 발생합니다. 이 기간 동안에는 이전 버전의 감지기 모델 인스턴스가 입력을 계속 처리할 수 있습니다. 이전 감지기 모델에서 정의한 경보를 계속 받을 수 있습니다. 업데이트를 다시 확인하거나 오류를 보고하기 전에 7분 이상 기다리십시오.

작업 트리거 실패(조건 충족 시)

조건이 충족되지만 감지기가 동작을 트리거하지 못하거나 새 상태로 전환하지 못합니다.

Solution

감지기 조건식의 평가 결과가 부울 값인지 확인하십시오. 결과가 부울 값이 아닌 경우 이 값은 `false`와 같으며 `action` 또는 이벤트에서 지정된 `nextState`로 전환을 트리거하지 않습니다. 자세한 정보는 조건 표현식 조건을 참조하십시오.

작업 트리거 실패(임계값 위반 시)

조건식의 변수가 지정된 값에 도달해도 감지기가 동작 또는 이벤트 전환을 트리거하지 않습니다.

Solution

`setVariable`을 `onInput`, `onEnter` 또는 `onExit`로 업데이트하면 현재 처리 주기 동안 `condition`을 평가할 때 새 값이 사용되지 않습니다. 대신 현재 주기가 완료될 때까지 원래 값이 사용됩니다. 감지기 모델 정의에서 `evaluationMethod` 파라미터를 설정하여 이 동작을 변경할 수 있습니다. `evaluationMethod`를 `SERIAL`로 설정하면 이벤트가 정의된 순서대로 변수가 업데이트되고 이벤트 조건이 평가됩니다. `evaluationMethod`를 `BATCH`(기본값)로 설정하면 변수가 업데이트되고 모든 이벤트 조건이 평가된 후에만 이벤트가 수행됩니다.

잘못된 상태 사용

`BatchPutMessage`를 사용하여 입력으로 메시지를 보내려고 하면 감지기가 잘못된 상태로 들어갑니다.

Solution

BatchPutMessage를 사용하여 여러 메시지를 입력으로 보내는 경우 메시지 또는 입력이 처리되는 순서가 보장되지 않습니다. 순서를 보장하려면 메시지를 한 번에 하나씩 보내고 성공이 확인될 때까지 매번 `BatchPutMessage` 동안 기다리십시오.

연결 메시지

API를 직간접적으로 호출하려고 하면 `('Connection aborted.', error(54, 'Connection reset by peer'))` 오류가 발생합니다.

Solution

OpenSSL이 TLS 1.1 이상 버전을 사용하여 연결을 설정하는지 확인하십시오. 대부분의 Linux 배포판이나 Windows 버전 7 이상에서는 이것이 기본값이어야 합니다. macOS 사용자는 OpenSSL을 업그레이드해야 할 수도 있습니다.

InvalidRequestException 메시지

`CreateDetectorModel` 및 `UpdateDetectorModel` API 호출을 시도하면 잘못된 요청 이상이 발생합니다.

Solution

다음에 따라 문제를 해결하십시오. 자세한 내용은CreateDetectorModel 및 UpdateDetectorModel을 참조하십시오.

- seconds와 durationExpression을 동시에 SetTimerAction의 파라미터로 사용하지 않도록 하십시오.
- durationExpression에 대한 문자열 표현식이 유효한지 확인하십시오. 문자열 표현식에는 숫자, 변수($variable.<variable-name>) 또는 입력 값($input.<input-name>.<path-to-datum>)이 포함될 수 있습니다.

Amazon CloudWatch Logs **action.setTimer** 오류

Amazon CloudWatch Logs를 설정하여 AWS IoT Events 감지기 모델 인스턴스를 모니터링할 수 있습니다. 다음은를 사용할 AWS IoT Events때에서 발생하는 일반적인 오류입니다action.setTimer.

- 오류: *<timer-name>*으로 이름이 지정된 타이머의 기간 표현식을 숫자로 평가할 수 없습니다.

  Solution

  durationExpression의 문자열 표현식을 숫자로 변환할 수 있는지 확인하십시오. 부울과 같은 다른 데이터 유형은 허용되지 않습니다.
- 오류: *<timer-name>*으로 이름이 지정된 타이머에 대한 기간 표현식의 평가 결과가 31622440보다 큽니다. 정확성을 보장하려면 지속 시간 표현식이 60~31622400 사이의 값을 참조해야 합니다.

  Solution

  타이머의 지속 시간이 31622400초 이하인지 확인하십시오. 지속 시간의 평가된 결과는 가장 가까운 정수로 내림됩니다.
- 오류: *<timer-name>*으로 이름이 지정된 타이머에 대한 지속 시간 표현식의 평가 결과가 60보다 작습니다. 정확성을 보장하려면 지속 시간 표현식이 60~31622400 사이의 값을 참조해야 합니다.

  Solution

  타이머의 지속 시간이 60초 이상인지 확인하십시오. 지속 시간의 평가된 결과는 가장 가까운 정수로 내림됩니다.
- 오류: *<timer-name>*으로 이름이 지정된 타이머의 지속 시간 표현식을 평가할 수 없습니다. 변수 이름, 입력 이름, 데이터 경로를 확인하여 기존 변수와 입력을 참조하는지 확인하십시오.

Solution

문자열 표현식이 기존 변수와 입력을 참조하는지 확인하십시오. 문자열 표현식에는 숫자, 변수(`$variable.`*`variable-name`*) 및 입력 값(`$input.`*`input-name`*`.`*`path-to-datum`*)이 포함될 수 있습니다.

- 오류: `<timer-name>` 이름의 타이머를 설정하지 못했습니다. 지속 시간 표현식을 확인한 후 다시 시도해 주십시오.

Solution

setTimerAction 작업을 참조하여 올바른 파라미터를 지정했는지 확인한 다음 타이머를 다시 설정하십시오.

자세한 내용은 AWS IoT Events 감지기 모델 개발 시 Amazon CloudWatch 로깅 활성화를 참조하세요.

Amazon CloudWatch 페이로드 오류

Amazon CloudWatch Logs를 설정하여 AWS IoT Events 감지기 모델 인스턴스를 모니터링할 수 있습니다. 다음은 작업 페이로드를 구성할 AWS IoT Events때에서 생성되는 일반적인 오류 및 경고입니다.

- 오류: 작업에 대한 표현식을 평가할 수 없습니다. 변수 이름, 입력 이름 및 데이터 경로가 기존 변수 및 입력 값을 참조하는지 확인하십시오. 또한 페이로드 크기가 최대 허용 크기인 1KB 미만인지 확인하십시오.

Solution

올바른 변수 이름, 입력 이름 및 데이터 경로를 입력했는지 확인하십시오. 작업 페이로드가 1KB보다 큰 경우에도 이 오류 메시지가 표시될 수 있습니다.

- 오류: *`<action-type>`*의 페이로드에 대한 콘텐츠 표현식을 파싱할 수 없습니다. 올바른 조건을 사용하여 콘텐츠 표현식을 입력합니다.

Solution

콘텐츠 표현식은 문자열(`'`*`string`*`'`), 변수(`$variable.`*`variable-name`*), 입력 값(`$input.`*`input-name`*`.`*`path-to-datum`*), 문자열 연결 및 `${}`를 포함하는 문자열을 포함할 수 있습니다.

- 오류: 페이로드 표현식 {*`expression`*}이 유효하지 않습니다. 정의된 페이로드 유형은 JSON이므로 문자열로 AWS IoT Events 평가할 표현식을 지정해야 합니다.

Solution

지정된 페이로드 유형이 JSON인 경우 AWS IoT Events 먼저 서비스가 표현식을 문자열로 평가할 수 있는지 확인합니다. 평가된 결과는 부울 또는 숫자일 수 없습니다. 검증에 실패하면 이 오류가 발생할 수 있습니다.

- 경고: 작업이 실행되었지만 작업 페이로드에 대한 콘텐츠 표현식을 유효한 JSON으로 평가할 수 없습니다. 정의된 페이로드 유형은 JSON입니다.

Solution

가 콘텐츠 표현식을 유효한 JSON으로 평가할 AWS IoT Events 수 없더라도 페이로드 유형을 로 정의하면 AWS IoT Events 가 JSON AWS IoT Events 작업 페이로드의 콘텐츠 표현식을 유효한 JSON으로 평가할 수 있는지 확인합니다.

자세한 내용은 AWS IoT Events 감지기 모델 개발 시 Amazon CloudWatch 로깅 활성화를 참조하세요.

호환되지 않는 데이터 유형

메시지: 다음 표현식 <reference>에 <expression>으로 호환되지 않는 데이터 유형 [<inferred-types>]이 있습니다.

Solution

다음과 같은 이유 중 하나로 오류가 발생할 수 있습니다.

- 참조의 평가 결과는 표현식의 다른 피연산자와 호환되지 않습니다.
- 함수에 전달된 인수 유형은 지원되지 않습니다.

표현식에서 참조를 사용하는 경우 다음을 확인하십시오.

- 하나 이상의 연산자와 함께 참조를 피연산자로 사용하는 경우, 참조하는 모든 데이터 유형이 호환되는지 확인하십시오.

  예를 들어, 다음 표현식에서 정수 2는 및 연산자 ==와 && 모두의 피연산자입니다. 피연산자를 호환하기 위해서는 $variable.testVariable + 1 및 $variable.testVariable이 정수 또는 10진수를 참조해야 합니다.

  또한 정수 1은(는) + 연산자의 피연산자입니다. 따라서 $variable.testVariable은(는) 정수 또는 10진수를 참조해야 합니다.

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

- 참조를 함수에 전달된 인수로 사용하는 경우, 함수가 참조하는 데이터 유형을 지원하는지 확인하십시오.

  예를 들어, 다음 `timeout("time-name")` 함수에는 큰따옴표가 있는 문자열이 인수로 필요합니다. `### ##` 값에 대한 참조를 사용하는 경우 큰따옴표가 있는 문자열을 참조해야 합니다.

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

> **Note**
>
> `convert(type, expression)` 함수의 경우 `##` 값에 대한 참조를 사용하는 경우 참조의 평가 결과는 `String`, `Decimal` 또는 `Boolean`이어야 합니다.

자세한 내용은 AWS IoT Events 표현식의 입력 및 변수에 대한 참조 단원을 참조하십시오.

에 메시지를 보내지 못했습니다. AWS IoT Events

메시지: Iot Events에 메시지를 보내지 못했습니다.

Solution

다음과 같은 이유로 오류가 발생할 수 있습니다.

- 입력 메시지 페이로드에 `Input attribute Key`가 포함되어 있지 않습니다.
- `Input attribute Key`가 입력 정의에 지정된 것과 동일한 JSON 경로에 있지 않습니다.
- 입력 메시지가 AWS IoT Events 입력에 정의된 스키마와 일치하지 않습니다.

> **Note**
>
> 다른 서비스에서 데이터 모으기를 할 때도 오류가 발생합니다.

Example

예를 들어 AWS IoT Core에서 AWS IoT 규칙은 다음 메시지와 함께 실패합니다. `Verify the Input Attribute key.`

이 문제를 해결하려면 입력 페이로드 메시지 스키마가 AWS IoT Events 입력 정의를 준수하고 `Input attribute Key` 위치가 일치하는지 확인합니다. 자세한 내용은 에서 모델에 대한 입력 생성 AWS IoT Events 단원을 참조하여 AWS IoT Events 입력을 정의하는 방법을 알아보십시오.

에서 분석을 실행하여 감지기 모델 문제 해결 AWS IoT Events

AWS IoT Events 는 감지기 모델에 입력 데이터를 보내지 않고도 감지기 모델을 분석하고 분석 결과를 생성할 수 있습니다.는이 섹션에 설명된 일련의 분석을 AWS IoT Events 수행하여 감지기 모델을 확인합니다. 이 고급 문제 해결 솔루션은 또한 심각도 수준 및 위치를 포함한 진단 정보를 요약하므로 감지기 모델의 잠재적 문제를 신속하게 찾아 수정할 수 있습니다. 감지기 모델의 진단 오류 유형 및 메시지에 대한 자세한 내용은 에 대한 감지기 모델 분석 및 진단 정보 AWS IoT Events을 참조하십시오.

AWS IoT Events 콘솔, API, AWS Command Line Interface (AWS CLI) 또는 AWS SDK를 사용하여 감지기 모델 분석의 진단 오류 메시지를 볼 수 있습니다.

> **Note**
>
> - 감지기 모델을 게시하려면 먼저 모든 오류를 수정해야 합니다.
> - 생산 환경에서 감지기 모델을 사용하기 전에 경고를 검토하고 필요한 조치를 취하는 것이 좋습니다. 그렇지 않으면 감지기 모델이 예상대로 작동하지 않을 수 있습니다.
> - `RUNNING` 상태에서 동시에 최대 10개의 분석을 수행할 수 있습니다.

감지기 모델을 분석하는 방법을 알아보려면 감지기 모델 분석 AWS IoT Events (콘솔) 또는 AWS IoT Events (AWS CLI)에서 감지기 모델 분석를 참조하십시오.

주제

- 에 대한 감지기 모델 분석 및 진단 정보 AWS IoT Events
- 감지기 모델 분석 AWS IoT Events (콘솔)
- AWS IoT Events (AWS CLI)에서 감지기 모델 분석

에 대한 감지기 모델 분석 및 진단 정보 AWS IoT Events

감지기 모델 분석은 다음과 같은 진단 정보를 수집합니다.

- 수준 — 분석 결과의 심각도 수준입니다. 심각도에 따라 분석 결과는 세 가지 일반적인 범주로 분류됩니다.
  - 정보(`INFO`) — 정보 결과는 감지기 모델의 중요한 필드에 대한 정보를 제공합니다. 이러한 유형의 결과에는 일반적으로 즉각적인 조치가 필요하지 않습니다.
  - 경고(`WARNING`) — 경고 결과는 감지기 모델에 문제를 일으킬 수 있는 분야에 특히 주의를 기울입니다. 생산 환경에서 감지기 모델을 사용하기 전에 경고를 검토하고 필요한 조치를 취하는 것이 좋습니다. 그렇지 않으면 감지기 모델이 예상대로 작동하지 않을 수 있습니다.
  - 오류(`ERROR`) — 오류 결과는 감지기 모델에서 발견된 문제에 대해 알려줍니다. AWS IoT Events는 감지기 모델을 게시하기 전에 이 분석 세트를 자동으로 수행합니다. 감지기 모델을 게시하려면 먼저 모든 오류를 수정해야 합니다.
- 위치 - 분석 결과가 참조하는 감지기 모델에서 필드를 찾는 데 사용할 수 있는 정보를 포함합니다. 위치에는 일반적으로 상태 이름, 전환 이벤트 이름, 이벤트 이름 및 표현식(예: `in state TemperatureCheck in onEnter in event Init in action setVariable`)이 포함됩니다.
- 유형 — 분석 결과의 유형입니다. 분석 유형은 다음과 같은 범주로 분류됩니다.
  - `supported-actions` - 지정된 이벤트 또는 전환 이벤트가 감지되면 작업을 호출할 AWS IoT Events 수 있습니다. 타이머를 사용하거나 변수를 설정하거나 다른 AWS 서비스로 데이터를 보내도록 기본 제공 작업을 정의할 수 있습니다. AWS 서비스를 사용할 수 있는 AWS 리전에서 다른 AWS 서비스와 함께 작동하는 작업을 지정해야 합니다.
  - `service-limits` - 한도라고도 하는 서비스 할당량은 AWS 계정의 최대 또는 최소 서비스 리소스 또는 작업 수입니다. 다르게 표시되지 않는 한, 리전별로 각 할당량이 적용됩니다. 비즈니스 요구 사항에 따라 감지기 모델을 업데이트하여 제한이 발생하지 않도록 하거나 할당량 증가를 요청할 수 있습니다. 일부 할당량에 대한 증가를 요청할 수 있으며 다른 할당량은 늘릴 수 없습니다. 자세한 내용은 할당량을 참조하십시오.
- **`structure`**— 감지기 모델은 상태와 같은 모든 필수 구성 요소를 포함해야 하며 AWS IoT Events가 지원하는 구조를 따라야 합니다. 감지기 모델에는 중요한 이벤트를 탐지하기 위해 들어오는 입력 데이터를 평가하는 상태와 조건이 하나 이상 있어야 합니다. 이벤트가 감지되면 감지기 모델은 다음 상태로 전환되어 작업을 호출할 수 있습니다. 이러한 이벤트를 전환 이벤트라고 합니다. 전환 이벤트는 진행할 다음 단계로 진행되어야 합니다.
- **`expression-syntax`**— AWS IoT Events 은(는) 감지기 모델을 생성하고 업데이트할 때 값을 지정하는 여러 가지 방법을 제공합니다. 표현식에서 리터럴, 연산자, 함수, 참조 및 대체 템플릿을 사용할

수 있습니다. 표현식을 사용하여 리터럴 값을 지정하거나 특정 값을 지정하기 전에 표현식을 평가할 AWS IoT Events 수 있습니다. 표현식은 필수 조건을 따라야 합니다. 자세한 내용은 이벤트 데이터를 필터링, 변환 및 처리하는 표현식 단원을 참조하십시오.

의 감지기 모델 표현식은 특정 데이터 또는 리소스를 참조할 AWS IoT Events 수 있습니다.

- **`data-type`** — AWS IoT Events 은(는) 정수, 십진수, 문자열 및 부울 데이터 유형을 지원합니다. 가 표현식 평가 중에 한 데이터 형식의 데이터를 다른 데이터 형식으로 자동 변환할 AWS IoT Events 수 있는 경우 이러한 데이터 형식은 호환됩니다.

  > **Note**
  >
  > - AWS IoT Events에서 지원하는 호환 가능한 데이터 유형은 정수와 10진수뿐입니다.
  > - AWS IoT Events 는 정수를 문자열로 변환할 수 없기 때문에 AWS IoT Events 산술 표현식을 평가할 수 없습니다.

- **`referenced-data`**— 데이터를 사용하려면 먼저 감지기 모델에서 참조되는 데이터를 정의해야 합니다. 예를 들어 DynamoDB 테이블로 데이터를 보내려면 먼저 테이블 이름을 참조하는 변수를 정의해야 표현식(`$variable.TableName`)에 변수를 사용할 수 있습니다.
- **`referenced-resource`**— 감지기 모델이 사용하는 리소스는 사용 가능해야 합니다. 리소스를 사용하려면 먼저 리소스를 정의해야 합니다. 예를 들어 온실의 온도를 모니터링하기 위한 감지기 모델을 생성하려고 합니다. `$input.TemperatureInput.sensorData.temperature`를 사용하여 온도를 참조하려면 먼저 들어오는 온도 데이터를 감지기 모델로 라우팅하는 입력(`$input.TemperatureInput`)을 정의해야 합니다.

오류를 해결하고 감지기 모델 분석을 통해 가능한 해결책을 찾으려면 다음 섹션을 참조하십시오.

에서 감지기 모델 오류 문제 해결 AWS IoT Events

위에서 설명한 오류 유형은 감지기 모델에 대한 진단 정보를 제공하며 검색할 수 있는 메시지에 해당합니다. 이 메시지와 제안된 솔루션을 사용하여 감지기 모델의 오류를 해결하십시오.

메시지 및 솔루션

- Location
- supported-actions
- service-limits
- structure

- expression-syntax
- data-type
- referenced-data
- referenced-resource

`Location`

`Location`에 대한 정보가 포함된 분석 결과는 다음 오류 메시지에 해당합니다.

- 메시지 — 분석 결과에 대한 추가 정보가 들어 있습니다. 정보, 경고 또는 오류 메시지일 수 있습니다.

  솔루션: AWS IoT Events 현재 지원되지 않는 작업을 지정한 경우 이 오류 메시지가 표시될 수 있습니다. 지원되는 작업 목록은 에서 데이터를 수신하고 작업을 트리거하는 데 지원되는 작업 AWS IoT Events을 참조하십시오.

`supported-actions`

`supported-actions`에 대한 정보가 포함된 분석 결과는 다음 오류 메시지에 해당합니다.

- 메시지: 작업 정의에 잘못된 작업 유형이 있습니다: *`## ##`*.

  솔루션: AWS IoT Events 현재 지원되지 않는 작업을 지정한 경우 이 오류 메시지가 표시될 수 있습니다. 지원되는 작업 목록은 에서 데이터를 수신하고 작업을 트리거하는 데 지원되는 작업 AWS IoT Events을 참조하십시오.
- 메시지: DetectorModel 정의에 *`aws-service`* 작업이 있지만 *`aws-service`*서비스가 리전 *`## ##`*에서 지원되지 않습니다.

  해결 방법: 지정한 작업이에서 지원되지 AWS IoT Events만 현재 리전에서 작업을 사용할 수 없는 경우이 오류 메시지가 표시될 수 있습니다. 이는 리전에서 사용할 수 없는 AWS 서비스로 데이터를 보내려고 할 때 발생할 수 있습니다. 또한 AWS IoT Events 및 사용 중인 서비스 모두에 대해 동일한 리전을 AWS 선택해야 합니다.

`service-limits`

`service-limits`에 대한 정보가 포함된 분석 결과는 다음 오류 메시지에 해당합니다.

- 메시지: 페이로드에 허용된 콘텐츠 표현식이 상태 *`## ##`*의 이벤트 *`### ##`*에 있는 *`content-expression-size`* 바이트 제한을 초과했습니다.

솔루션: 작업 페이로드의 콘텐츠 표현식이 1,024 바이트를 초과하는 경우 이 오류 메시지가 표시될 수 있습니다. 페이로드의 콘텐츠 표현식 크기는 최대 1,024 바이트입니다.

- 메시지: 감지기 모델 정의에 허용된 상태 수가 *states-per-detector-model* 제한을 초과했습니다.

  솔루션: 감지기 모델의 상태가 20개를 초과하는 경우 이 오류 메시지가 표시될 수 있습니다. 감지기 모델은 최대 20개의 상태를 보유할 수 있습니다.

- 메시지: 타이머 *### ##*의 지속 시간은 최소*minimum-timer-duration*초 이상이어야 합니다.

  솔루션: 타이머 지속 시간이 60초 미만인 경우 이 오류 메시지가 표시될 수 있습니다. 타이머 지속 시간은 60초에서 31,622,400초 사이로 설정하는 것이 좋습니다. 타이머 지속 시간의 표현식을 지정하는 경우 지속 시간 표현식의 평가된 결과는 가장 가까운 정수로 내림됩니다.

- 메시지: 이벤트당 허용된 액션 수가 감지기 모델 정의의 *actions-per-event* 제한을 초과했습니다.

  솔루션: 이벤트에 10개 이상의 작업이 있는 경우 이 오류 메시지가 표시될 수 있습니다. 감지기 모델의 각 이벤트에 대해 최대 10개의 동작을 보유할 수 있습니다.

- 메시지: 상태당 허용되는 전환 이벤트 수가 감지기 모델 정의의 *transition-events-per-state* 제한을 초과했습니다.

  솔루션: 상태에 20개가 넘는 전환 이벤트가 있는 경우 이 오류 메시지가 표시될 수 있습니다. 감지기 모델의 각 상태에 대해 최대 20개의 전환 이벤트를 보유할 수 있습니다.

- 메시지: 상태당 허용된 이벤트 수가 감지기 모델 정의의 *events-per-state*제한을 초과했습니다.

  솔루션: 상태에 20개가 넘는 이벤트가 있는 경우 이 오류 메시지가 표시될 수 있습니다. 감지기 모델의 각 상태에 대해 최대 20개의 이벤트를 보유할 수 있습니다.

- 메시지: 단일 입력과 연결할 수 있는 최대 감지기 모델 수를 초과했습니다. 입력 *input-name*은 감지기 모델별 *detector-models-per-input* 경로에 사용됩니다.

  솔루션: 10개 이상의 감지기 모델로 입력을 라우팅하려고 하면 이 경고 메시지가 표시될 수 있습니다. 단일 감지기 모델에 최대 10개의 서로 다른 감지기 모델을 연결할 수 있습니다.

structure

structure에 대한 정보가 포함된 분석 결과는 다음 오류 메시지에 해당합니다.

- 메시지: 작업에는 한 가지 유형만 정의될 수 있지만 *number-of-types* 유형이 많은 작업이 있습니다. 별도의 작업으로 분리해 주십시오.

  솔루션: API 작업을 사용하여 감지기 모델을 생성하거나 업데이트하여 단일 필드에 두 개 이상의 작업을 지정한 경우 이 오류 메시지가 표시될 수 있습니다. `Action` 객체 배열을 정의할 수 있습니다. 각 작업을 별도의 객체로 정의해야 합니다.
- 메시지: TransitionEvent *transition-event-name*는 존재하지 않는 상태 *state-name*로 전환됩니다.

  해결 방법: AWS IoT Events 에서 전환 이벤트가 참조한 다음 상태를 찾을 수 없는 경우이 오류 메시지가 표시될 수 있습니다. 다음 상태가 정의되어 있고 올바른 상태 이름을 입력했는지 확인하십시오.
- 메시지: DetectorModelDefinition에 공유된 상태 이름이 있습니다. *number-of-states* 반복이 있는 상태 *state-name*을 찾았습니다.

  솔루션: 하나 이상의 상태에 같은 이름을 사용하는 경우 이 오류 메시지가 표시될 수 있습니다. 감지기 모델의 각 상태에 고유한 이름을 지정해야 합니다. 이름은 1~128자여야 합니다. 유효한 문자는 a-z, A-Z, 0-9, _(밑줄) 및 -(하이픈)입니다.
- 메시지: 정의의 initialStateName *initial-state-name*이 정의된 상태와 일치하지 않습니다.

  솔루션: 초기 상태 이름이 잘못된 경우 이 오류 메시지가 표시될 수 있습니다. 감지기 모델은 입력이 도착할 때까지 초기(시작) 상태를 유지합니다. 입력이 도착하면 감지기 모델은 즉시 다음 상태로 전환됩니다. 초기 상태 이름이 정의된 상태의 이름이고 올바른 이름을 입력했는지 확인하십시오.
- 메시지: 감지기 모델 정의는 조건에서 하나 이상의 입력을 사용해야 합니다.

  솔루션: 조건에 입력값을 지정하지 않은 경우 이 오류가 발생할 수 있습니다. 하나 이상의 조건에서 하나 이상의 입력을 사용해야 합니다. 그렇지 않으면 수신 데이터를 평가 AWS IoT Events 하지 않습니다.
- 메시지: SetTimer에서는 초와 지속 시간 표현식 중 하나만 설정할 수 있습니다.

  솔루션: 타이머에 `seconds`와 `durationExpression`을 둘 다 사용한 경우 이 오류 메시지가 표시될 수 있습니다. `seconds` 및 `durationExpression`, 둘 중 하나를 `SetTimerAction`의 파라미터로 사용해야 합니다. 자세한 내용을 알아보려면 AWS IoT Events API 참조의 SetTimerAction를 참조하십시오.
- 메시지: 감지기 모델의 동작에 접근할 수 없습니다. 작업을 시작하는 조건을 확인하십시오.

  솔루션: 감지기 모델의 동작에 도달할 수 없는 경우 이벤트 조건은 false로 평가됩니다. 동작이 포함된 이벤트의 상태를 확인하여 해당 동작이 true로 평가되는지 확인하십시오. 이벤트 조건이 true로 평가되면 조치를 수행할 수 있어야 합니다.

- 메시지: 입력 속성을 읽는 중이지만 타이머 만료로 인한 것일 수 있습니다.

  솔루션: 다음 중 하나가 발생할 경우 입력 속성 값을 읽을 수 있습니다.

  - 새 입력 값을 받았습니다.
  - 감지기의 타이머가 만료된 경우.

  입력 속성이 해당 입력에 대한 새 값이 수신될 때만 평가되도록 하려면 다음과 같이 조건에 `triggerType("Message")` 함수 호출을 포함시키십시오.

  감지기 모델에서 평가 중인 원래 조건:

  ```
  if ($input.HeartBeat.status == "OFFLINE")
  ```

  이는 다음과 비슷합니다.

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

  조건에 제공된 초기 입력보다 `triggerType("Message")` 함수 호출이 먼저 오는 경우. 이 기법을 사용하면 `triggerType("Message")` 함수가 true로 평가되어 새 입력값을 받는 조건을 만족하게 됩니다. `triggerType` 함수 사용에 대한 자세한 내용은 AWS IoT Events 개발자 안내서의 표현식 섹션에서 `triggerType`을 참조하십시오.
- 메시지: 감지기 모델의 상태에 도달할 수 없습니다. 원하는 상태로 전환되는 조건을 확인하십시오.

  솔루션: 감지기 모델의 상태에 도달할 수 없는 경우 해당 상태로의 전환을 유도하는 조건이 false로 평가됩니다. 감지기 모델에서 도달할 수 없는 상태로의 수신 전환 조건이 true로 평가되는지 확인하여 원하는 상태에 도달할 수 있도록 하십시오.
- 메시지: 타이머가 만료되면 예상치 못한 양의 메시지가 전송될 수 있습니다.

  솔루션: 타이머가 만료되어 감지기 모델이 예상치 못한 양의 메시지를 보내는 무한 상태에 빠지는 것을 방지하려면 다음과 같이 감지기 모델의 조건에서 `triggerType("Message")` 함수 호출을 사용하는 것이 좋습니다.

  감지기 모델에서 평가 중인 원래 조건:

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

  다음과 비슷한 조건으로 변환됩니다.

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

조건에 제공된 초기 입력보다 `triggerType("Message")` 함수 호출이 먼저 오는 경우.

이 변경으로 감지기에서 타이머 동작이 시작되지 않아 메시지가 무한 루프 전송되는 것을 방지할 수 있습니다. 감지기에서 타이머 액션을 사용하는 방법에 대한 자세한 내용은 AWS IoT Events 개발자 가이드의 기본 제공 작업 사용 페이지를 참조하십시오.

expression-syntax

`expression-syntax`에 대한 정보가 포함된 분석 결과는 다음 오류 메시지에 해당합니다.

- 메시지: 페이로드 표현식{*expression*}이 유효하지 않습니다. 정의된 페이로드 유형은 JSON이므로 문자열로 AWS IoT Events 평가할 표현식을 지정해야 합니다.

  해결 방법: 지정된 페이로드 유형이 JSON인 경우 AWS IoT Events 먼저 서비스가 표현식을 문자열로 평가할 수 있는지 확인합니다. 평가된 결과는 부울 또는 숫자일 수 없습니다. 검증에 실패하면 이 오류가 발생할 수 있습니다.
- 메시지: `SetVariableAction.value`은(는) 표현식이어야 합니다. '*variable-value*' 값을 파싱하지 못했습니다.

  솔루션: `SetVariableAction`을 사용하여 `name` 및 `value` 변수를 정의할 수 있습니다. `value`은(는) 문자열, 숫자 또는 부울 값일 수 있습니다. `value`에 대해 표현식을 지정할 수도 있습니다. 자세한 내용을 알아보려면 AWS IoT Events API 참조의 SetVariableAction을 참조하십시오.
- 메시지: DynamoDB 작업에 대한 속성(*attribute-name*) 표현식을 파싱할 수 없습니다. 표현식을 올바른 조건으로 입력합니다.

  솔루션: 대체 템플릿 `DynamoDBAction`의 모든 파라미터에 표현식을 사용해야 합니다. 자세한 내용은 AWS IoT Events API 참조의 DynamoDBAction을 참조하십시오.
- 메시지: DynamoDBv2 작업에 대한 테이블 이름 표현식을 파싱할 수 없습니다. 표현식을 올바른 조건으로 입력합니다.

  솔루션: `DynamoDBv2Action`의 `tableName`은(는) 문자열이어야 합니다. `tableName`에는 표현식을 사용해야 합니다. 표현식에서 리터럴, 연산자, 함수, 참조 및 대체 템플릿을 사용할 수 있습니다. 자세한 내용은 AWS IoT Events API 참조의 DynamoDBv2Action을 참조하십시오.
- 메시지: 표현식을 유효한 JSON으로 평가할 수 없습니다. DynamoDBv2 작업은 JSON 페이로드 유형만 지원합니다.

솔루션: DynamoDBv2의 페이로드 유형은 JSON이어야 합니다. 가 페이로드의 콘텐츠 표현식을 유효한 JSON으로 AWS IoT Events 평가할 수 있는지 확인합니다. 자세한 내용은 AWS IoT Events API 참조에서 DynamoDBv2Action을 참조하십시오.

- 메시지: *action-type*의 페이로드에 대한 콘텐츠 표현식을 파싱할 수 없습니다. 올바른 조건을 사용하여 콘텐츠 표현식을 입력합니다.

  솔루션: 콘텐츠 표현식에는 문자열('*string*'), 변수($variable.*variable-name*), 입력 값($input.*input-name*.*path-to-datum*), 문자열 연결 및 ${}를 포함하는 문자열이 포함될 수 있습니다.

- 메시지: 사용자 지정된 페이로드는 비어 있지 않아야 합니다.

  해결 방법: 작업에 대해 사용자 지정 페이로드를 선택하고 AWS IoT Events 콘솔에 콘텐츠 표현식을 입력하지 않은 경우이 오류 메시지가 표시될 수 있습니다. 사용자 지정 페이로드를 선택하는 경우 사용자 지정 페이로드에 콘텐츠 표현식을 입력해야 합니다. 자세한 내용은 AWS IoT Events API 참조의 Payload를 참조하십시오.

- 메시지: 타이머 '*timer-name*'에 대한 기간 표현식 '*duration-expression*'을 파싱하지 못했습니다.

  솔루션: 타이머에 대한 지속 시간 표현식의 평가 결과는 60~31622400 사이의 값이어야 합니다. 지속 시간의 평가된 결과는 가장 가까운 정수로 내림됩니다.

- 메시지: *action-name*에 대한 표현식 '*expression*'을 파싱하지 못했습니다.

  솔루션: 지정된 작업의 표현식 조건이 잘못된 경우 이 메시지가 표시될 수 있습니다. 표현식을 올바른 조건으로 입력했는지 확인하십시오. 자세한 내용은 에서 디바이스 데이터를 필터링하고 작업을 정의하는 구문 AWS IoT Events 섹션을 참조하십시오.

- 메시지: IotSitewiseAction에 대한 *fieldName*을 파싱할 수 없습니다. 표현식에 올바른 조건을 사용해야 합니다.

  해결 방법:가에 대한 *fieldName*을 구문 분석할 수 AWS IoT Events 없는 경우이 오류가 발생할 수 있습니다IotSitewiseAction. *FieldName*에서 AWS IoT Events 가 파싱할 수 있는 표현식을 사용하는지 확인하십시오. 자세한 내용은 AWS IoT Events API 참조의 IotSiteWiseAction을 참조하십시오.

data-type

data-type에 대한 정보가 포함된 분석 결과는 다음 오류 메시지에 해당합니다.

- 메시지: 타이머 *timer-name* 대한 기간 표현식 *duration-expression*이 유효하지 않으므로 숫자를 반환해야 합니다.

  해결 방법: AWS IoT Events 가 타이머의 기간 표현식을 숫자로 평가할 수 없는 경우이 오류 메시지가 표시될 수 있습니다. durationExpression가 숫자로 변환될 수 있는지 확인하십시오. 부울과 같은 다른 데이터 유형은 지원되지 않습니다.

- 메시지: 표현식 *condition-expression*이 유효한 조건 표현식이 아닙니다.

  해결 방법:가를 부울 값으로 평가할 수 없는 경우이 오류 메시지가 표시될 condition-expression 수 AWS IoT Events 있습니다. 부울 값은 TRUE 또는 FALSE 중 하나여야 합니다. 조건 표현식이 부울 값으로 변환될 수 있는지 확인하십시오. 결과가 부울 값이 아닌 경우 해당 값은 FALSE와 동일하며 이벤트에서 nextState 지정된 값으로 전환하거나 액션을 호출하지 않습니다.

- 메시지: 다음 표현식 *expression*에서 *reference*로 사용할 수 없는 데이터 유형[*inferred-type*]이 발견되었습니다.

  솔루션: 솔루션: 감지기 모델에서 동일한 입력 속성 또는 변수에 대한 모든 표현식은 동일한 데이터 유형을 참조해야 합니다.

  다음 정보를 활용하여 문제를 해결합니다.

  - 하나 이상의 연산자와 함께 참조를 피연산자로 사용하는 경우, 참조하는 모든 데이터 유형이 호환되는지 확인하십시오.

    예를 들어, 다음 표현식에서 정수 2는 및 연산자 ==와 && 모두의 피연산자입니다. 피연산자를 호환하기 위해서는 $variable.testVariable + 1 및 $variable.testVariable이 정수 또는 10진수를 참조해야 합니다.

    또한 정수 1은(는) + 연산자의 피연산자입니다. 따라서 $variable.testVariable은(는) 정수 또는 10진수를 참조해야 합니다.

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

  - 참조를 함수에 전달된 인수로 사용하는 경우, 함수가 참조하는 데이터 유형을 지원하는지 확인하십시오.

    예를 들어, 다음 timeout("*time-name*") 함수에는 큰따옴표가 있는 문자열이 인수로 필요합니다. *### ##* 값에 대한 참조를 사용하는 경우 큰따옴표가 있는 문자열을 참조해야 합니다.

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

> **Note**
>
> `convert(type, expression)` 함수의 경우 `##` 값에 대한 참조를 사용하는 경우 참조의 평가 결과는 `String`, `Decimal` 또는 `Boolean`이어야 합니다.

자세한 내용은 AWS IoT Events 표현식의 입력 및 변수에 대한 참조 섹션을 참조하십시오.

- 메시지: *reference*로 사용할 수 없는 데이터 유형 [*inferred-types*]입니다. 이로 인해 런타임 오류가 발생할 수 있습니다.

  솔루션: 동일한 입력 속성 또는 변수에 대한 두 표현식이 두 데이터 유형을 참조하는 경우 이 경고 메시지가 표시될 수 있습니다. 동일한 입력 속성이나 변수에 대한 표현식이 감지기 모델의 동일한 데이터 유형을 참조하는지 확인하십시오.

- 메시지: 연산자[*operator*]에 입력한 데이터 유형[*inferred-types*]이 다음 표현식 '*expression*'과 호환되지 않습니다.

  솔루션: 표현식이 지정된 연산자와 호환되지 않는 데이터 유형을 결합한 경우 이 오류 메시지가 나타날 수 있습니다. 예를 들어 다음 표현식에서 연산자 +은(는) 정수, 10진수 및 문자열 데이터 형식과 호환되지만 부울 데이터 유형의 피연산자와는 호환되지 않습니다.

  ```
  true + false
  ```

  연산자와 함께 사용하는 데이터 유형이 호환되는지 확인해야 합니다.

- 메시지: *input-attribute*에 대해 발견된 데이터 유형[*infered-types*]이 호환되지 않아 런타임 오류가 발생할 수 있습니다.

  솔루션: 동일한 입력 속성에 대한 두 표현식이 상태의 `OnEnterLifecycle`, 또는 상태의 `OnInputLifecycle`과 `OnExitLifecycle` 모두에 대해 두 데이터 유형을 참조하는 경우 이 오류 메시지가 표시될 수 있습니다. 감지기 모델의 각 상태에 대해 `OnEnterLifecycle`(또는 `OnInputLifecycle` 및 `OnExitLifecycle` 둘 다)의 표현식이 동일한 데이터 유형을 참조하는지 확인하십시오.

- 메시지: 페이로드 표현식[*expression*]이 유효하지 않습니다. 페이로드 유형이 JSON 형식이므로 런타임 시 문자열로 평가되는 식을 지정하십시오.

  해결 방법: 지정된 페이로드 유형이 JSON이지만 표현식을 문자열로 평가할 AWS IoT Events 수 없는 경우이 오류가 발생할 수 있습니다. 평가 결과가 부울이나 숫자가 아닌 문자열인지 확인하십시오.

- 메시지: 보간 표현식{*interpolated-expression*}은 런타임 시 정수 또는 부울 값으로 평가되어야 합니다. 그렇지 않으면 페이로드 표현식 {*payload-expression*}을 런타임에 유효한 JSON으로 파싱할 수 없습니다.

  해결 방법: AWS IoT Events 가 보간된 표현식을 정수 또는 부울 값으로 평가할 수 없는 경우이 오류 메시지가 표시될 수 있습니다. 문자열과 같은 다른 데이터 유형은 지원되지 않으므로 보간된 식을 정수 또는 부울 값으로 변환할 수 있는지 확인하십시오.

- 메시지: `IotSitewiseAction` 필드 *expression*의 표현식 유형은 유형 *defined-type*으로 정의되고 유형 *inferred-type*으로 추론됩니다. 정의된 유형과 유추된 유형은 동일해야 합니다.

  솔루션: `IotSitewiseAction`의 `propertyValue`에서 표현식 데이터 유형이 유추된 AWS IoT Events의 데이터 유형과 다르게 정의된 경우 이 오류 메시지가 표시될 수 있습니다. 감지기 모델에서 이 표현식의 모든 인스턴스에 동일한 데이터 유형을 사용해야 합니다.

- 메시지: `setTimer` 작업에 사용되는 데이터 유형 [*inferred-types*]은 다음 표현식 *expression*에서는 `Integer`로 평가되지 않습니다.

  솔루션: 기간 표현식의 유추된 데이터 유형이 정수 또는 10진수가 아닌 경우 이 오류 메시지가 표시될 수 있습니다. `durationExpression`가 숫자로 변환할 수 있는지 확인하십시오. 부울과 문자열과 같은 기타 데이터 유형은 지원되지 않습니다.

- 메시지: 비교 연산자[*operator*]의 피연산자와 함께 사용되는 데이터 유형 [*interred-types*]은 다음 표현식 *expression*에서 호환되지 않습니다.

  솔루션: 감지기 모델의 조건식(*###*) 에 있는 연산자의 *####*에 대해 유추된 데이터 유형이 일치하지 않습니다. 피연산자는 감지기 모델의 다른 모든 부분에서 일치하는 데이터 유형과 함께 사용해야 합니다.

> **Tip**
>
> `convert`을 사용하여 감지기 모델에서 표현식의 데이터 유형을 변경할 수 있습니다. 자세한 내용은 AWS IoT Events 표현식에 사용할 함수 섹션을 참조하십시오.

`referenced-data`

`referenced-data`에 대한 정보가 포함된 분석 결과는 다음 오류 메시지에 해당합니다.

- 메시지: 손상된 타이머 감지: 타이머 *timer-name*이 표현식에 사용되었지만 설정되지 않았습니다.

솔루션: 설정되지 않은 타이머를 사용하는 경우 이 오류 메시지가 표시될 수 있습니다. 표현식에 사용하기 전에 타이머를 설정해야 합니다. 또한 타이머 이름을 정확하게 입력해야 합니다.

- 메시지: 손상된 변수 감지: 변수 *variable-name*이 표현식에 사용되었지만 설정되지 않았습니다.

  솔루션: 설정되지 않은 변수를 사용하는 경우 이 오류 메시지가 표시될 수 있습니다. 표현식에서 변수를 사용하려면 먼저 변수를 설정해야 합니다. 또한 올바른 변수 이름을 입력해야 합니다.

- 메시지: 손상된 변수 감지: 변수가 값으로 설정되기 전에 표현식에 사용됩니다.

  솔루션: 표현식에서 변수를 평가하려면 먼저 각 변수를 값에 할당해야 합니다. 변수 값을 검색할 수 있도록 사용하기 전에 매번 변수 값을 설정하십시오. 또한 올바른 변수 이름을 입력해야 합니다.

`referenced-resource`

`referenced-resource`에 대한 정보가 포함된 분석 결과는 다음 오류 메시지에 해당합니다.

- 메시지: 감지기 모델 정의에는 존재하지 않는 입력에 대한 참조가 포함되어 있습니다.

  솔루션: 표현식을 사용하여 존재하지 않는 입력을 참조하는 경우 이 오류 메시지가 표시될 수 있습니다. 표현식이 기존 입력을 참조하는지 확인하고 올바른 입력 이름을 입력하십시오. 입력이 없는 경우에는 먼저 하나를 생성합니다.

- 메시지: 감지기 모델 정의에 잘못된 InputName *input-name*이 포함되어 있습니다.

  솔루션: 감지기 모델에 잘못된 입력 이름이 포함된 경우 이 오류 메시지가 표시될 수 있습니다. 올바른 입력 이름을 입력해야 합니다. 입력 이름은 1~128자여야 합니다. 유효한 문자는 a-z, A-Z, 0-9, _(밑줄) 및 -(하이픈)입니다.

감지기 모델 분석 AWS IoT Events (콘솔)

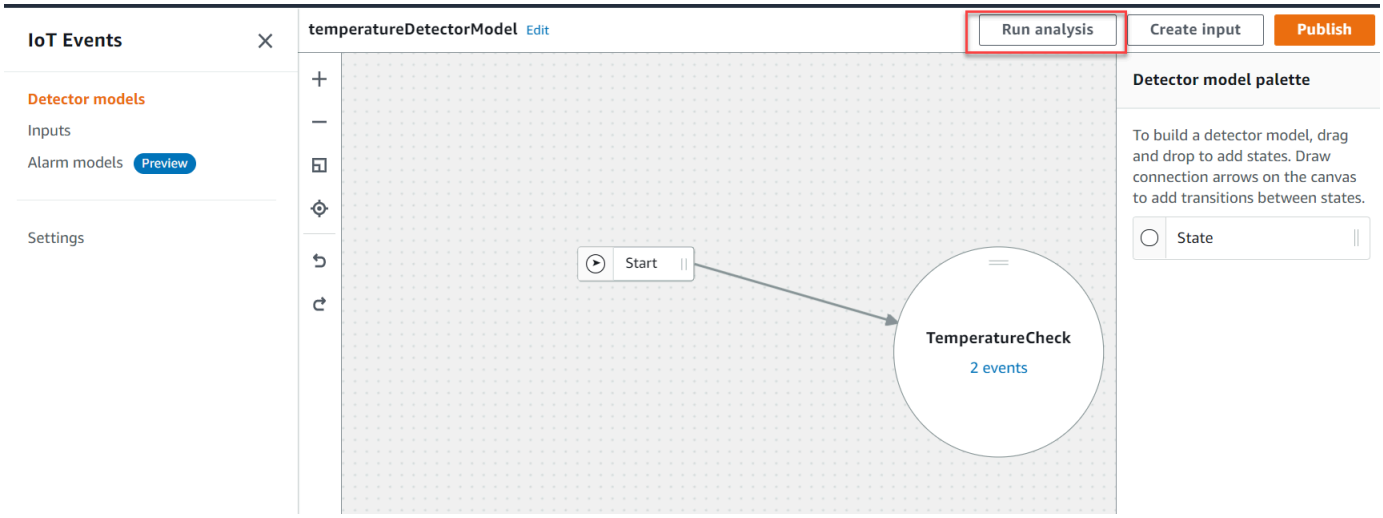
AWS IoT Events 를 사용하면 이벤트를 감지하고 AWS IoT Events API로 작업을 트리거하여 IoT 데이터를 모니터링하고 이에 대응할 수 있습니다. 다음 단계에서는 AWS IoT Events 콘솔을 사용하여 감지기 모델을 분석합니다.

> **Note**
>
> 가 감지기 모델 분석을 AWS IoT Events 시작한 후 최대 24시간 동안 분석 결과를 검색할 수 있습니다.

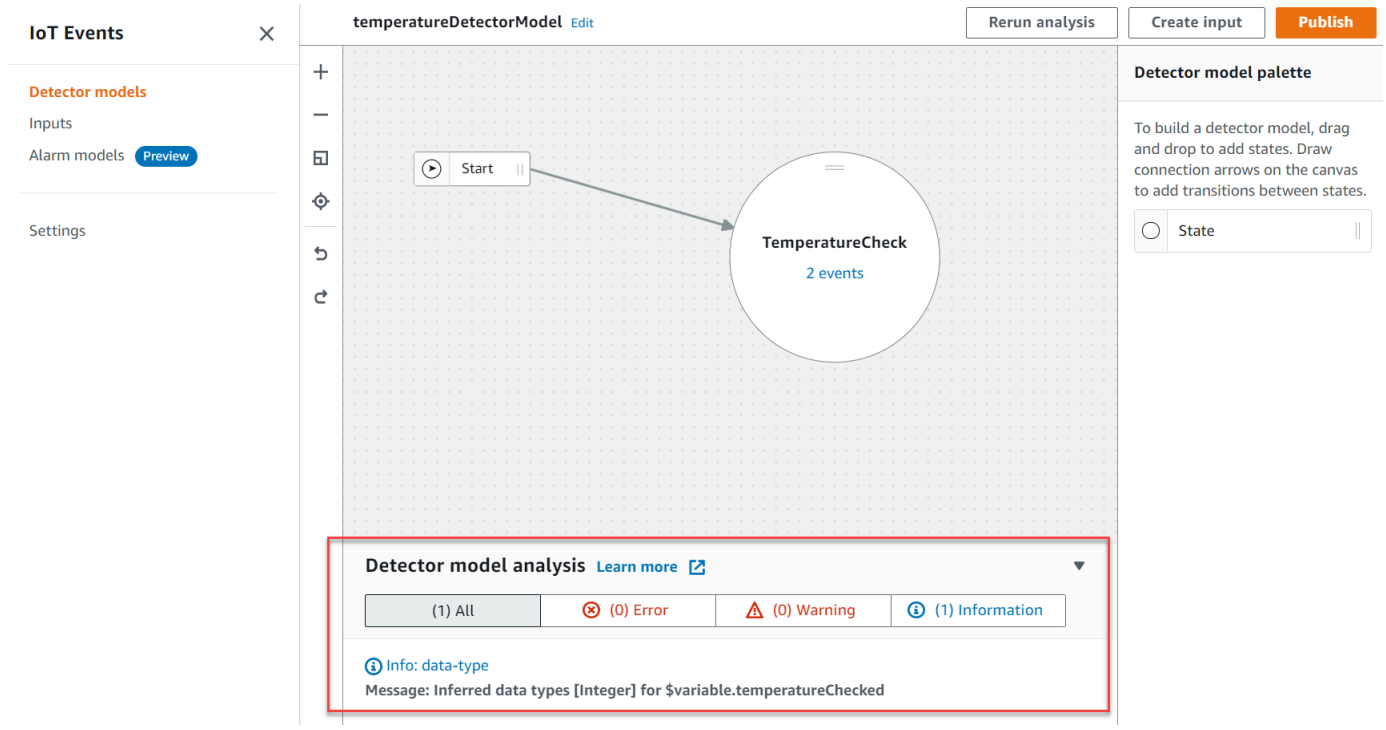
감지기 모델 분석은 모델을 최적화하고, 잠재적 문제를 식별하고, 의도한 대로 작동하는지 확인하는 데 도움이 될 수 있습니다. 예를 들어 풍력 발전소에서 감지기 모델 분석은 모델이 비정상적인 진동 패턴을 기반으로 잠재적 기어 장애를 올바르게 식별하는지 여부를 확인할 수 있습니다. 또는 풍속이 안전한 작동 임계값을 초과할 때 모델이 유지 관리 알림을 정확하게 트리거하는 경우. 분석을 기반으로 모델을 개선하면 예측 유지 관리를 개선하고, 가동 중지 시간을 줄이고, 전반적인 에너지 생산 효율성을 높일 수 있습니다.

감지기 모델을 분석하려면

1. AWS IoT Events 콘솔에 로그인합니다.
2. 탐색 창에서 (감지기 모델)를 선택합니다.
3. 감지기 모델에서 대상 감지기 모델을 선택합니다.
4. 감지기 모델 페이지에서 편집을 선택합니다.
5. 오른쪽 상단 모서리에서 분석 실행을 선택합니다.



다음은 AWS IoT Events 콘솔의 예제 분석 결과입니다.



AWS IoT Events (AWS CLI)에서 감지기 모델 분석

AWS IoT Events 감지기 모델을 프로그래밍 방식으로 분석하면 구조, 동작 및 성능에 대한 귀중한 인사이트를 얻을 수 있습니다. 이 API 기반 접근 방식을 사용하면 자동화된 분석, 기존 워크플로와의 통합, 여러 감지기 모델에서 대량 작업을 수행할 수 있습니다. StartDetectorModelAnalysis API를 활용하면 모델에 대한 심층 검사를 시작하여 잠재적 문제를 식별하고, 로직 흐름을 최적화하고, IoT 이벤트 처리가 비즈니스 요구 사항에 부합하는지 확인할 수 있습니다.

다음 단계에서는 AWS CLI 를 사용하여 감지기 모델을 분석합니다.

를 사용하여 감지기 모델을 분석하려면 AWS CLI

1. 다음 명령을 실행해 분석을 시작합니다.

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

> **Note**
>
> *file-name*을 감지기 모델 정의가 포함된 파일의 이름으로 대체합니다.

Example감지기 모델 정의

```
{
    "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"
    }
}
```

AWS CLI 를 사용하여 기존 감지기 모델을 분석하는 경우 다음 중 하나를 선택하여 감지기 모델 정의를 검색합니다.

- AWS IoT Events 콘솔을 사용하려면 다음을 수행합니다.
  1. 탐색 창에서 감지기 모델을 선택합니다.
  2. 감지기 모델에서 대상 감지기 모델을 선택합니다.
  3. 작업에서 감지기 모델 내보내기를 선택하여 감지기 모델을 다운로드합니다. 감지기 모델은 JSON으로 저장됩니다.
  4. 감지기 모델 JSON 파일을 엽니다.
  5. `detectorModelDefinition` 객체만 있으면 됩니다. 다음을 제거하십시오.
     - 페이지 상단의 첫 번째 중괄호({)
     - `detectorModel` 라인
     - `detectorModelConfiguration` 객체
     - 페이지 하단의 마지막 중괄호(})
  6. 파일을 저장합니다.
- 를 사용하려면 다음을 AWS CLI수행합니다.
  1. 터미널에서 다음 명령을 실행하십시오.

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

  2. *detector-model-name*을 감지기 모델 이름으로 바꾸십시오.
  3. `detectorModelDefinition` 개체를 텍스트 편집기에 복사합니다.
  4. `detectorModelDefinition` 외부에 중괄호({})를 추가합니다.
  5. 파일을 JSON으로 저장합니다.

Example응답의 예

```
{
```

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

2. 출력에서 분석 ID를 복사합니다.
3. 다음 명령을 실행하여 분석 상태를 검색합니다.

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

> **Note**
>
> *analysis-id*를 복사한 분석 ID로 바꾸십시오.

Example응답의 예

```
{
    "status": "COMPLETE"
}
```

상태는 다음 값 중 하나일 수 있습니다.

- RUNNING - 감지 AWS IoT Events 기 모델을 분석합니다. 이 프로세스가 완료되는 데 최대 1분이 걸릴 수 있습니다.
- COMPLETE - 감지기 모델 분석을 AWS IoT Events 완료했습니다.
- FAILED –감지기 모델을 분석할 수 AWS IoT Events 없습니다. 나중에 다시 시도해 주세요.

4. 다음 명령을 실행하여 감지기 모델의 분석 결과를 하나 이상 검색합니다.

> **Note**
>
> *analysis-id*를 복사한 분석 ID로 바꾸십시오.

```
aws iotevents get-detector-model-analysis-results --analysis-id "analysis-id"
```

Example응답의 예

```
{
```

```
    "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**
>
> 가 감지기 모델 분석을 AWS IoT Events 시작한 후 최대 24시간 동안 분석 결과를 검색할 수 있습니다.

AWS IoT Events 명령

이 장에서는에서 사용할 수 있는 모든 API 작업에 대한 포괄적인 가이드를 제공합니다 AWS IoT Events. 지원되는 웹 서비스 프로토콜에서 각 작업에 대한 샘플 요청, 응답 및 잠재적 오류를 포함하여 자세한 설명을 제공합니다. 이러한 API 작업을 이해하면 IoT 애플리케이션에 효과적으로 통합하고 이벤트 감지 및 응답 워크플로를 자동화 AWS IoT Events 할 수 있습니다.

AWS IoT Events 작업

AWS IoT Events API 명령을 사용하여 입력 및 감지기 모델을 생성, 읽기, 업데이트 및 삭제하고 해당 버전을 나열할 수 있습니다. 자세한 내용은 AWS IoT Events API 참조의 AWS IoT Events 에서 지원하는 작업 및 데이터 유형을 참조하세요.

AWS CLI 명령 참조의 AWS IoT Events 섹션에는 관리 및 조작에 사용할 수 있는 AWS CLI 명령이 포함되어 있습니다 AWS IoT Events.

AWS IoT Events 데이터

AWS IoT Events 데이터 API 명령을 사용하여 감지기에 입력을 보내고, 감지기를 나열하고, 감지기의 상태를 보거나 업데이트할 수 있습니다. 자세한 내용은 AWS IoT Events API 참조의 데이터에서 지원하는 작업 및 AWS IoT Events 데이터 유형을 참조하세요.

AWS CLI 명령 참조의 AWS IoT Events 데이터 섹션에는 AWS IoT Events 데이터를 처리하는 데 사용할 수 있는 AWS CLI 명령이 포함되어 있습니다.

에 대한 문서 기록 AWS IoT Events

아래 표에 2020년 9월 17일 이후 AWS IoT Events 개발자 안내서의 주요 변경 사항이 설명되어 있습니다. 이 설명서에 대한 업데이트 알림을 더 자세히 받으려면 RSS 피드를 구독하면 됩니다.

변경 사항	설명	날짜
지원 종료 알림	지원 종료 공지: 2026년 5월 20일에는에 대한 지원을 중단할 AWS 예정입니다 AWS IoT Events. 2026년 5월 20일 이후에는 AWS IoT Events 콘솔 또는 AWS IoT Events 리소스에 더 이상 액세스할 수 없습니다.	2025년 5월 20일
리전 출시	AWS IoT Events 이제 아시아 태평양(뭄바이) 리전에서를 사용할 수 있습니다.	2021년 9월 30일
리전 출시	AWS IoT Events 이제를 AWS GovCloud(미국 서부) 리전에서 사용할 수 있습니다.	2021년 9월 22일
분석을 실행하여 감지기 모델 문제 해결	AWS IoT Events 는 이제 감지기 모델을 분석하고 감지기 모델 문제를 해결하는 데 사용할 수 있는 분석 결과를 생성할 수 있습니다.	2021년 2월 23일
리전 출시	중국(베이징) AWS IoT Events 에서 출시되었습니다.	2020년 9월 30일
표현식 사용	표현식 작성 방법을 보여주는 예제가 추가되었습니다.	2020년 9월 22일
경보로 모니터링	경보를 사용하면 데이터의 변경 사항을 모니터링할 수 있습니다. 임계값 위반 시 알림을 보	2020년 6월 1일

	내는 경보를 생성할 수 있습니다.	

이전 업데이트

다음 표에서는 2020년 9월 18일 이전 AWS IoT Events 개발자 안내서의 주요 변경 사항이 설명되어 있습니다.

변경 사항	설명	날짜
표현식 참조에 유형 검증 추가	표현식 참조에 유형 검증 정보가 추가되었습니다.	2020년 8월 3일
다른 서비스에 대한 리전 경고 추가	AWS IoT Events 및 기타 AWS 서비스에 대해 동일한 리전을 선택하는 것과 관련된 경고가 추가되었습니다.	2020년 5월 7일
추가, 업데이트	• 페이로드 사용자 지정 기능 • 새 이벤트 작업: Amazon DynamoDB 및 AWS IoT SiteWise	2020년 4월 27일
감지기 모델 조건 표현식에 대한 내장 함수 추가	감지기 모델 조건식에 대한 내장 함수가 추가되었습니다.	2019년 9월 10일
감지기 모델 예제 추가	감지기 모델에 대한 예제가 추가되었습니다.	2019년 8월 5일
새 이벤트 작업 추가	다음에 대한 새 이벤트 작업이 추가되었습니다. • Lambda • Amazon SQS • Kinesis Data Firehose • AWS IoT Events 입력	2019년 7월 19일

변경 사항	설명	날짜
추가, 수정	• `timeout()` 함수에 대한 설명을 업데이트했습니다. • 계정 비활성에 관한 모범 사례가 추가됨.	2019년 6월 11일
업데이트된 권한 정책 및 콘솔 디버그 옵션	• 콘솔 권한 정책이 업데이트되었습니다. • 콘솔 디버그 옵션 페이지 이미지가 업데이트되었습니다.	2019년 6월 5일
업데이트	AWS IoT Events 서비스는 일반 가용성에 개방됩니다.	2019년 5월 30일
추가, 업데이트	• 보안 정보가 업데이트되었습니다. • 주석이 달린 감지기 모델 예제가 추가되었습니다.	2019년 5월 22일
예제 및 필수 권한 추가	Amazon SNS 페이로드 예제를 추가하고에 필요한 권한을 추가했습니다`CreateDetectorModel` .	2019년 5월 17일
추가 보안 정보 추가	보안 섹션에 정보가 추가되었습니다.	2019년 5월 9일
제한된 미리 보기 출시	제한된 미리보기 출시에 대한 설명서.	2019년 3월 28일