aws

Guide du développeur

AWS IoT Events

AWS IoT Events: Guide du développeur

Table of Contents

..... viii
Qu'est-ce que c'est AWS IoT Events ? 1
Avantages et fonctionnalités 1
Cas d’utilisation 3
Surveiller et entretenir les appareils distants 3
Gérez les robots industriels 3
Systèmes d'automatisation des bâtiments sur rails 3
AWS IoT Events fin du support 4
Considérations à prendre en compte lors de la migration hors de AWS IoT Events 4
Modèles de détecteur 5
Comparaison des architectures 6
Étape 1 : Exporter les configurations des modèles de AWS IoT Events détecteurs (facultatif) 7
Étape 2 : Créer un rôle IAM 8
Étape 3 : Création d'Amazon Kinesis Data Streams 10
Étape 4 : Création ou mise à jour de la règle de routage des messages MQTT 11
Étape 5 : Obtenir le point de terminaison pour le sujet MQTT de destination 13
Étape 6 : Création d'une table Amazon DynamoDB 13
Étape 7 : Création d'une AWS Lambda fonction (console) 14
Étape 8 : ajouter un déclencheur Amazon Kinesis Data Streams 22
Étape 9 : Tester les fonctionnalités d'ingestion et de sortie des données (AWS CLI) 23
Alertes 24
Comparaison des architectures 24
Étape 1 : activer les notifications MQTT sur la propriété de l'actif 25
Étape 2 : Création d'une AWS Lambda fonction 26
Étape 3 : Création d'une règle de routage des AWS IoT Core messages 28
Étape 4 : Afficher les CloudWatch métriques 28
Étape 5 : créer des CloudWatch alarmes 29
Étape 6 : (Facultatif) importez l' CloudWatch alarme dans AWS IoT SiteWise 29
Configuration 30
Configuration d'un Compte AWS 30
Inscrivez-vous pour un Compte AWS 30
Création d’un utilisateur doté d’un accès administratif 31
Configuration des autorisations pour AWS IoT Events 32

Autorisations d'action 33
Sécurisation des données d'entrée 35
Politique relative aux rôles de CloudWatch journalisation d'Amazon 36
Politique relative aux rôles de messagerie Amazon SNS 38
Prise en main 40
Conditions préalables 42
Création d'une entrée 43
Création d'un fichier d'entrée JSON 43
Création et configuration d'une entrée 43
Création d'une entrée dans le modèle de détecteur 44
Création d'un modèle de détecteur 45
Testez le modèle de détecteur 52
Bonnes pratiques 56
Activer la CloudWatch journalisation Amazon lors du développement AWS IoT Events de modèles de détecteurs 56
Publiez régulièrement pour enregistrer votre modèle de détecteur lorsque vous travaillez dans la AWS IoT Events console 57
Tutoriels 58
Utilisation AWS IoT Events pour surveiller vos appareils IoT 58
Comment savoir de quels états vous avez besoin dans un modèle de détecteur ? 60
Comment savoir si vous avez besoin d'une ou de plusieurs instances de détecteur ? 62
step-by-stepExemple simple 62
Création d'une entrée pour capturer les données de l'appareil 65
Création d'un modèle de détecteur pour représenter les états des appareils 66
Envoyer des messages sous forme d'entrées à un détecteur 70
Restrictions et limites du modèle de détecteur 73
Un exemple commenté : le contrôle de la température HVAC 77
Définitions d'entrée pour les modèles de détecteurs 78
Création d'une définition de modèle de détecteur 81
Utilisation des BatchUpdateDetector 102
Utilisation BatchPutMessage pour les entrées 104
Ingérer des messages MQTT 106
Génération de messages Amazon SNS 108
Configuration de l' DescribeDetector API 109
Utiliser le moteur de AWS IoT Core règles 111
Actions prises en charge 115

Utiliser des actions intégrées 116
Régler l'action du chronomètre 116
Action de réinitialisation du chronomètre 117
Effacer l'action du chronomètre 117
Définir une action variable 117
Collaborez avec d'autres AWS services 118
AWS IoT Core 119
AWS IoT Events 120
AWS IoT SiteWise 121
Amazon DynamoDB 124
Amazon DynamoDB (version 2) 126
Amazon Data Firehose 127
AWS Lambda 129
Amazon Simple Notification Service 130
Amazon Simple Queue Service 131
Expressions 133
Syntaxe pour filtrer les données de l'appareil 133
Littéraux 133
Opérateurs 134
Fonctions pour les expressions 135
Référence pour les entrées et les variables dans les expressions 140
Modèles de substitution 143
Usage 143
Écrire AWS IoT Events des expressions 144
Exemples de modèles de détecteurs 146
Régulation de la température HVAC 146
Histoire de fond 146
Définitions d'entrée 147
Définition du modèle de détecteur 150
BatchPutMessage exemples 167
BatchUpdateDetector exemple 173
AWS IoT Core moteur de règles 175
Grues 178
Envoyer des commandes 179
Modèles de détecteur 181
Inputs 188

Messages 188
Exemple : détection d'événements à l'aide de capteurs 190
Appareil HeartBeat 192
Alarme ISA 195
Alarme simple 205
Surveillance à l’aide des alarmes 210
Travailler avec AWS IoT SiteWise 210
Reconnaître le flux 211
Création d'un modèle d'alarme 211
Exigences 212
Création d'un modèle d'alarme (console) 212
Répondre aux alarmes 216
Gestion des notifications d'alarme 217
Création d’une fonction Lambda 217
Utilisation de la fonction Lambda 227
Gérer les destinataires des alarmes 228
Sécurité 229
Gestion des identités et des accès 230
Public ciblé 230
Authentification par des identités 230
Gestion de l’accès à l’aide de politiques 232
En savoir plus sur la gestion des identités et des accès 233
Comment AWS IoT Events fonctionne avec IAM 233
Exemples de politiques basées sur l’identité 238
Prévention interservices confuse des adjoints pour AWS IoT Events 244
Résolution des problèmes 249
Contrôle 251
Outils disponibles pour surveiller AWS IoT Events 252
Surveillance AWS IoT Events avec Amazon CloudWatch 253
Journalisation des appels d' AWS IoT Events API avec AWS CloudTrail 255
Validation de conformité 274
Résilience 275
Sécurité de l’infrastructure 275
Quotas 276
Identification 277
Principes de base des étiquettes 277

Limites et restrictions liées aux balises 278
Utilisation des balises avec des politiques IAM 278
Résolution des problèmes 282
AWS IoT Events Problèmes courants et solutions 282
Erreurs de création du modèle de détecteur 283
Mises à jour depuis un modèle de détecteur supprimé 283
Défaillance du déclencheur d'action (en cas de respect d'une condition) 283
Défaillance du déclencheur de l'action (en cas de dépassement d'un seuil) 284
Utilisation incorrecte de l'état 284
Message de connexion 284
InvalidRequestException message 285
`action.setTimer`Erreurs Amazon CloudWatch Logs 285
Erreurs de CloudWatch charge utile Amazon 286
Types de données incompatibles 288
Impossible d'envoyer le message à AWS IoT Events 289
Dépannage d'un modèle de détecteur 290
Informations diagnostiques 291
Analyser un modèle de détecteur (console) 305
Analyser un modèle de détecteur (AWS CLI) 306
Commandes 312
AWS IoT Events actions 312
AWS IoT Events données 312
Historique de la documentation 313
Mises à jour antérieures 314

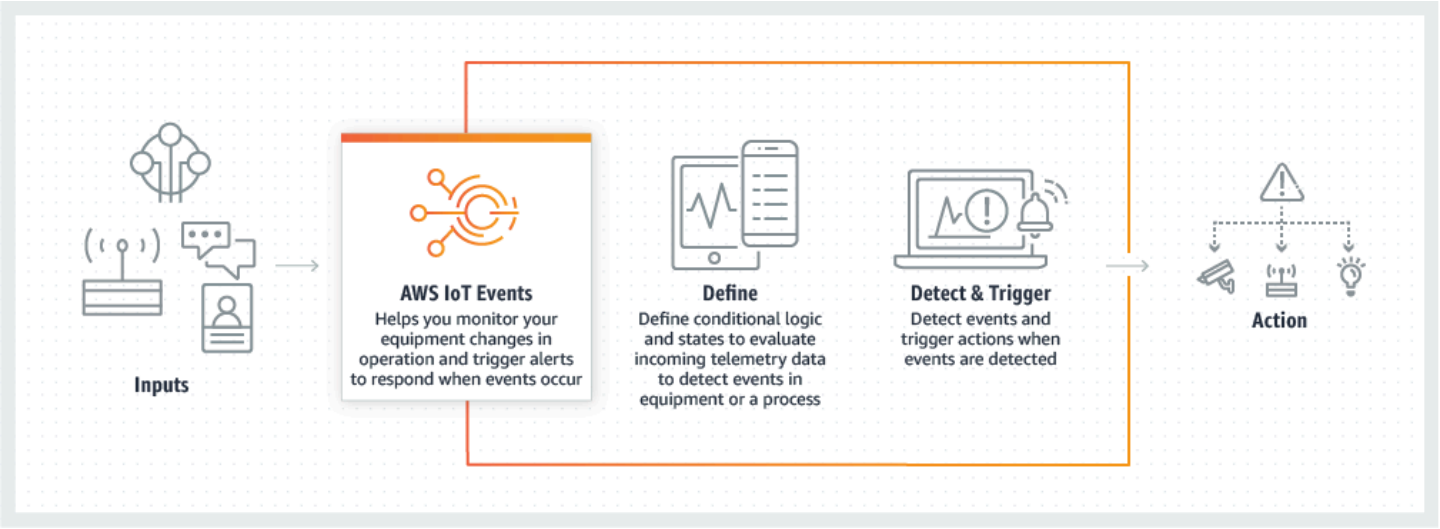
Avis de fin de support : le 20 mai 2026, AWS le support de AWS IoT Events. Après le 20 mai 2026, vous ne pourrez plus accéder à la AWS IoT Events console ni aux AWS IoT Events ressources. Pour plus d'informations, consultez AWS IoT Events la fin du support.

Les traductions sont fournies par des outils de traduction automatique. En cas de conflit entre le contenu d'une traduction et celui de la version originale en anglais, la version anglaise prévaudra.

Qu'est-ce que c'est AWS IoT Events ?

AWS IoT Events vous permet de surveiller votre parc d'équipements ou d'appareils pour détecter les pannes ou les changements de fonctionnement, et de déclencher des actions lorsque de tels événements se produisent. AWS IoT Events surveille en permanence les données des capteurs IoT provenant des appareils, des processus, des applications et d'autres AWS services afin d'identifier les événements importants afin que vous puissiez agir.

AWS IoT Events Utilisez-le pour créer des applications complexes de surveillance des événements dans le AWS cloud auxquelles vous pouvez accéder via la AWS IoT Events console ou APIs.



Rubriques

- Avantages et fonctionnalités
- Cas d’utilisation

Avantages et fonctionnalités

Accepter les entrées provenant de plusieurs sources

AWS IoT Events accepte les entrées provenant de nombreuses sources de données de télémétrie IoT. Il s'agit notamment des capteurs, des applications de gestion et d'autres AWS IoT services, tels que AWS IoT Core et AWS IoT Analytics. Vous pouvez transférer n'importe quelle entrée de données de télémétrie à l'aide AWS IoT Events d'une interface API (`BatchPutMessage`API) standard ou de la AWS IoT Events console.

Pour plus d'informations sur la prise en main AWS IoT Events, consultezCommencer à utiliser la AWS IoT Events console.

Utilisez des expressions logiques simples pour reconnaître des modèles d'événements complexes

AWS IoT Events peut reconnaître des modèles d'événements impliquant plusieurs entrées provenant d'un seul appareil ou d'une seule application IoT, ou de divers équipements et de nombreux capteurs indépendants. Cela est particulièrement utile car chaque capteur et chaque application fournissent des informations importantes. Mais ce n'est qu'en combinant diverses données de capteurs et d'applications que vous pouvez obtenir une image complète des performances et de la qualité des opérations. Vous pouvez configurer les AWS IoT Events détecteurs pour qu'ils reconnaissent ces événements à l'aide d'expressions logiques simples au lieu d'un code complexe.

Pour plus d'informations sur les expressions logiques, consultezExpressions pour filtrer, transformer et traiter les données d'événements.

Déclenchez des actions en fonction des événements

AWS IoT Events vous permet de déclencher directement des actions dans Amazon Simple Notification Service (Amazon SNS), Lambda AWS IoT Core, Amazon SQS et Amazon Kinesis Firehose. Vous pouvez également déclencher une AWS Lambda fonction à l'aide du moteur de AWS IoT règles qui permet d'effectuer des actions à l'aide d'autres services, tels qu'Amazon Connect, ou de vos propres applications de planification des ressources d'entreprise (ERP).

AWS IoT Events inclut une bibliothèque prédéfinie d'actions que vous pouvez entreprendre et vous permet également de définir les vôtres.

Pour en savoir plus sur le déclenchement d'actions en fonction d'événements, consultezActions prises en charge pour recevoir des données et déclencher des actions dans AWS IoT Events.

Évoluez automatiquement pour répondre aux exigences de votre flotte

AWS IoT Events redimensionne automatiquement lorsque vous connectez des appareils homogènes. Vous pouvez définir un détecteur une seule fois pour un type d'appareil spécifique, et le service adaptera et gérera automatiquement toutes les instances de cet appareil auxquelles il est connecté AWS IoT Events.

Pour découvrir des exemples de modèles de détecteurs, voirAWS IoT Events exemples de modèles de détecteurs.

Cas d’utilisation

AWS IoT Events a de nombreuses utilisations. Voici quelques exemples de cas d'utilisation.

Surveiller et entretenir les appareils distants

La surveillance d'un parc de machines déployées à distance peut s'avérer difficile, en particulier lorsqu'un dysfonctionnement survient sans contexte clair. Si une machine cesse de fonctionner, cela peut impliquer le remplacement de l'ensemble de l'unité de traitement ou de la machine. Mais cela n'est pas durable. AWS IoT Events Vous pouvez ainsi recevoir des messages provenant de plusieurs capteurs installés sur chaque machine pour vous aider à diagnostiquer des problèmes spécifiques au fil du temps. Au lieu de remplacer l'ensemble de l'unité, vous disposez désormais des informations nécessaires pour envoyer à un technicien la pièce exacte à remplacer. Avec des millions de machines, les économies peuvent atteindre des millions de dollars, réduisant ainsi le coût total de possession ou d'entretien de chaque machine.

Gérez les robots industriels

Le déploiement de robots dans vos installations pour automatiser le mouvement des colis peut améliorer considérablement l'efficacité. Pour minimiser les coûts, les robots peuvent être équipés de capteurs simples et peu coûteux qui transmettent les données au cloud. Cependant, avec des dizaines de capteurs et des centaines de modes de fonctionnement, il peut être difficile de détecter les problèmes en temps réel. Vous pouvez ainsi créer un système expert qui traite les données de ces capteurs dans le cloud, en créant des alertes pour avertir automatiquement le personnel technique en cas de panne imminente. AWS IoT Events

Systèmes d'automatisation des bâtiments sur rails

Dans les centres de données, la surveillance des températures élevées et du faible taux d'humidité permet d'éviter les pannes d'équipement. Les capteurs sont souvent achetés auprès de nombreux fabricants et chaque type est livré avec son propre logiciel de gestion. Cependant, les logiciels de gestion de différents fournisseurs ne sont parfois pas compatibles, ce qui complique la détection des problèmes. Vous pouvez ainsi configurer des alertes pour informer vos analystes des opérations des problèmes liés à vos systèmes de chauffage et de refroidissement bien avant les pannes. AWS IoT Events De cette façon, vous pouvez éviter un arrêt imprévu du centre de données qui coûterait des milliers de dollars en remplacement de l'équipement et pourrait entraîner une perte de revenus.

AWS IoT Events fin du support

Après mûre réflexion, nous avons décidé de mettre fin au support du AWS IoT Events service à compter du 20 mai 2026. AWS IoT Events n'acceptera plus de nouveaux clients à compter du 20 mai 2025. En tant que client existant disposant d'un compte inscrit au service avant le 20 mai 2025, vous pouvez continuer à utiliser les AWS IoT Events fonctionnalités. Après le 20 mai 2026, vous ne pourrez plus utiliser AWS IoT Events.

Cette page fournit des instructions et des éléments à prendre en compte pour aider les AWS IoT Events clients à passer à une autre solution répondant aux besoins de votre entreprise.

> **Note**
>
> Les solutions présentées dans ces guides sont destinées à servir d'exemples illustratifs, et non à remplacer des fonctionnalités prêtes à être produites. AWS IoT Events Personnalisez le code, le flux de travail et les AWS ressources connexes en fonction des besoins de votre entreprise.

Rubriques

- Considérations à prendre en compte lors de la migration hors de AWS IoT Events
- Procédure de migration pour les modèles de détecteurs dans AWS IoT Events
- Procédure de migration pour les AWS IoT SiteWise alarmes dans AWS IoT Events

Considérations à prendre en compte lors de la migration hors de AWS IoT Events

- Mettez en œuvre les meilleures pratiques de sécurité, notamment en utilisant des rôles IAM dotés de privilèges minimaux pour chaque composant et en chiffrant les données au repos et en transit. Pour plus d'informations, consultez Bonnes pratiques de sécurité dans IAM dans le Guide de l'utilisateur IAM.
- Tenez compte du nombre de partitions pour le flux Kinesis en fonction de vos exigences en matière d'ingestion de données. Pour plus d'informations sur les partitions Kinesis, consultez la terminologie et les concepts relatifs à Amazon Kinesis Data Streams dans le manuel du développeur Amazon Kinesis Data Streams.

- Configurez une surveillance et un débogage complets à l'aide CloudWatch des métriques et des journaux. Pour plus d'informations, voir Qu'est-ce que c'est CloudWatch ? dans le guide de CloudWatch l'utilisateur Amazon.
- Réfléchissez à la structure de votre gestion des erreurs, notamment à la manière de gérer les messages dont le traitement échoue à plusieurs reprises, à la mise en œuvre de politiques de nouvelle tentative et à la mise en place d'un processus pour isoler et analyser les messages problématiques.
- Utilisez le calculateur de AWS prix pour estimer les coûts correspondant à votre cas d'utilisation spécifique.

Procédure de migration pour les modèles de détecteurs dans AWS IoT Events

Cette section décrit les solutions alternatives qui fournissent des fonctionnalités de modèle de détecteur similaires à celles que vous utilisez lors de la migration AWS IoT Events.

Vous pouvez migrer l'ingestion de données via des AWS IoT Core règles vers une combinaison d'autres AWS services. Au lieu d'être ingérées via l'BatchPutMessageAPI, les données peuvent être acheminées vers le sujet AWS IoT Core MQTT.

Cette approche de migration utilise les sujets AWS IoT Core MQTT comme point d'entrée pour vos données IoT, en remplacement de la saisie directe dans. AWS IoT Events Les sujets MQTT sont choisis pour plusieurs raisons principales. Ils offrent une large compatibilité avec les appareils IoT en raison de l'utilisation généralisée du MQTT dans l'industrie. Ces rubriques peuvent traiter de gros volumes de messages provenant de nombreux appareils, ce qui garantit l'évolutivité. Ils offrent également une flexibilité dans le routage et le filtrage des messages en fonction du contenu ou du type d'appareil. De plus, les rubriques AWS IoT Core MQTT s'intègrent parfaitement aux autres AWS services, ce qui facilite le processus de migration.

Les données issues de sujets MQTT circulent vers une architecture combinant Amazon Kinesis Data Streams, AWS Lambda une fonction, une table Amazon DynamoDB et des plannings Amazon. EventBridge Cette combinaison de services reproduit et améliore les fonctionnalités précédemment fournies par AWS IoT Events, vous offrant ainsi plus de flexibilité et de contrôle sur votre pipeline de traitement des données IoT.

Comparaison des architectures

L' AWS IoT Events architecture actuelle ingère les données par le biais d'une AWS IoT Core règle et de l'`BatchPutMessage`API. Cette architecture est utilisée AWS IoT Core pour l'ingestion de données et la publication d'événements, les messages étant acheminés via des AWS IoT Events entrées vers des modèles de détecteurs qui définissent la logique de l'état. Un rôle IAM gère les autorisations nécessaires.

La nouvelle solution prend en charge AWS IoT Core l'ingestion de données (désormais avec des rubriques MQTT dédiées aux entrées et aux sorties). Il introduit Kinesis Data Streams pour le partitionnement des données et une fonction Lambda d'évaluation pour la logique des états. Les états des appareils sont désormais stockés dans une table DynamoDB, et un rôle IAM amélioré gère les autorisations sur ces services.

Objectif	Solution	Différences
Ingestion de données — Reçoit les données des appareils IoT	AWS IoT Core	Nécessite désormais deux rubriques MQTT distinctes : l'une pour l'ingestion des données du périphérique et l'autre pour la publication des événements de sortie
Orientation des messages — Achemine les messages entrants vers les services appropriés	AWS IoT Core règle de routage des messages	Conserve les mêmes fonctionnalités de routage, mais dirige désormais les messages vers Kinesis Data Streams au lieu de AWS IoT Events
Traitement des données — Gère et organise les flux de données entrants	Kinesis Data Streams	Remplace la fonctionnalité de AWS IoT Events saisie, permettant l'ingestion des données avec le partitionnement des identifiants des appareils pour le traitement des messages
Évaluation logique — Traite les changements d'état et déclenche des actions	Évaluateur Lambda	Remplace le modèle AWS IoT Events de détecteur, fournissant une évaluation de la logique d'état personnalisable par le biais du code au lieu d'un flux de travail visuel

Objectif	Solution	Différences
Gestion des états — Maintient l'état des appareils	Tableau DynamoDB	Nouveau composant qui fournit un stockage permanent des états des appareils, en remplacement de la gestion interne de AWS IoT Events l'état
Sécurité — Gère les autorisations de service	Rôle IAM	Les autorisations mises à jour incluent désormais l'accès à Kinesis Data Streams, à DynamoDB, EventBridge ainsi qu'aux autorisat ions existantes AWS IoT Core

Étape 1 : Exporter les configurations des modèles de AWS IoT Events détecteurs (facultatif)

Avant de créer de nouvelles ressources, exportez les définitions AWS IoT Events de vos modèles de détecteurs. Ils contiennent votre logique de traitement des événements et peuvent servir de référence historique pour la mise en œuvre de votre nouvelle solution.

Console

À l'aide du AWS IoT Events AWS Management Console, effectuez les étapes suivantes pour exporter les configurations de votre modèle de détecteur :

Pour exporter des modèles de détecteurs à l'aide du AWS Management Console

1. Connectez-vous à la console AWS IoT Events.
2. Dans le panneau de navigation de gauche, choisissez Modèles de détecteur.
3. Sélectionnez le modèle de détecteur à exporter.
4. Cliquez sur Exporter. Lisez le message d'information concernant la sortie, puis sélectionnez à nouveau Exporter.
5. Répétez le processus pour chaque modèle de détecteur que vous souhaitez exporter.

Un fichier contenant une sortie JSON de votre modèle de détecteur est ajouté au dossier de téléchargement de votre navigateur. Vous pouvez éventuellement enregistrer la configuration de chaque modèle de détecteur afin de préserver les données historiques.

AWS CLI

À l'aide de AWS CLI, exécutez les commandes suivantes pour exporter les configurations de votre modèle de détecteur :

Pour exporter des modèles de détecteurs à l'aide de AWS CLI

1. Répertoriez tous les modèles de détecteurs de votre compte :

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

2. Pour chaque modèle de détecteur, exportez sa configuration en exécutant :

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

3. Enregistrez la sortie pour chaque modèle de détecteur.

Étape 2 : Créer un rôle IAM

Créez un rôle IAM pour fournir des autorisations permettant de répliquer les fonctionnalités de. AWS IoT Events Dans cet exemple, le rôle donne accès à DynamoDB pour la gestion des états, pour la planification EventBridge, à Kinesis Data Streams pour l' AWS IoT Core ingestion de données, pour la publication de messages et pour la journalisation. CloudWatch Ensemble, ces services remplaceront AWS IoT Events.

1. Créez un rôle IAM avec les autorisations suivantes. Pour des instructions plus détaillées sur la création d'un rôle IAM, consultez la section Créer un rôle pour déléguer des autorisations à un AWS service dans le Guide de l'utilisateur IAM.

   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. Ajoutez la politique de confiance des rôles IAM suivante. Une politique de confiance permet aux AWS services spécifiés d'assumer le rôle IAM afin qu'ils puissent effectuer les actions nécessaires. Pour des instructions plus détaillées sur la création d'une politique de confiance IAM, voir Création d'un rôle à l'aide de politiques de confiance personnalisées dans le Guide de l'utilisateur IAM.

   JSON

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

Étape 3 : Création d'Amazon Kinesis Data Streams

Créez Amazon Kinesis Data Streams à l'aide AWS Management Console du AWS CLI ou.

Console

Pour créer un flux de données Kinesis à l'aide de AWS Management Console, suivez la procédure décrite sur la page Créer un flux de données du manuel Amazon Kinesis Data Streams Developer Guide.

Ajustez le nombre de partitions en fonction du nombre d'appareils et de la taille de la charge utile des messages.

AWS CLI

Utilisez AWS CLI et créez Amazon Kinesis Data Streams pour ingérer et partitionner les données de vos appareils.

Les Kinesis Data Streams sont utilisés dans cette migration pour remplacer la fonctionnalité AWS IoT Events d'ingestion de données de. Il fournit un moyen évolutif et efficace de collecter, traiter et analyser les données de streaming en temps réel à partir de vos appareils IoT, tout en offrant une gestion flexible des données et une intégration avec d'autres AWS services.

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

Ajustez le nombre de partitions en fonction du nombre d'appareils et de la taille de la charge utile des messages.

Étape 4 : Création ou mise à jour de la règle de routage des messages MQTT

Vous pouvez créer une nouvelle règle de routage des messages MQTT ou mettre à jour une règle existante.

Console

1. Déterminez si vous avez besoin d'une nouvelle règle de routage des messages MQTT ou si vous pouvez mettre à jour une règle existante.
2. Ouvrez la AWS IoT Core console.
3. Dans le volet de navigation, choisissez Message Routing, puis Rules.

4. Dans la section Gérer, choisissez Routage des messages, puis Règles.
5. Choisissez Créer une règle.
6. Sur la page Spécifier les propriétés de la règle, entrez le nom de la AWS IoT Core règle pour Nom de la règle. Dans le champ Description de la règle (facultatif), entrez une description pour indiquer que vous traitez des événements et que vous les transmettez à Kinesis Data Streams.
7. Sur la page Configurer l'instruction SQL, entrez ce qui suit pour l'instruction SQL :**`SELECT * FROM 'your-database'`**, puis choisissez Next.
8. Sur la page Associer des actions aux règles, et sous Actions des règles, sélectionnez kinesis.
9. Choisissez votre flux Kinesis pour le stream. Pour la clé de partition, saisissez **`your-instance-id`**. Sélectionnez le rôle approprié pour le rôle IAM, puis choisissez Ajouter une action de règle.

Pour plus d'informations, consultez la section Création de règles AWS IoT pour acheminer les données des appareils vers d'autres services.

AWS CLI

1. Créez un fichier JSON avec le contenu suivant. Ce fichier de configuration JSON définit une AWS IoT Core règle qui sélectionne tous les messages d'un sujet et les transmet au flux Kinesis spécifié, en utilisant l'ID d'instance comme clé de partition.

```
{
    "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. Créez la règle du sujet MQTT à l'aide du AWS CLI. Cette étape utilise le AWS CLI pour créer une règle de AWS IoT Core rubrique en utilisant la configuration définie dans le `events_rule.json` fichier.

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

Étape 5 : Obtenir le point de terminaison pour le sujet MQTT de destination

Utilisez la rubrique MQTT de destination pour configurer l'endroit où vos rubriques publient les messages sortants, en remplacement de la fonctionnalité précédemment gérée par AWS IoT Events. Le point de terminaison est propre à votre AWS compte et à votre région.

Console

1. Ouvrez la AWS IoT Core console.
2. Dans la section Connect du panneau de navigation de gauche, choisissez Configuration du domaine.
3. Choisissez la configuration du domaine IoT:Data-ATS pour ouvrir la page détaillée de la configuration.
4. Copiez la valeur du nom de domaine. Cette valeur est le point final. Enregistrez la valeur du point de terminaison, car vous en aurez besoin ultérieurement.

AWS CLI

Exécutez la commande suivante pour obtenir le AWS IoT Core point de terminaison permettant de publier les messages sortants pour votre compte.

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

Étape 6 : Création d'une table Amazon DynamoDB

Une table Amazon DynamoDB remplace la fonctionnalité de gestion d'état AWS IoT Events de, fournissant un moyen évolutif et flexible de conserver et de gérer l'état de vos appareils ainsi que la logique du modèle de détecteur dans votre nouvelle architecture de solution.

Console

Créez une table Amazon DynamoDB pour conserver l'état des modèles de détecteurs. Pour plus d'informations, consultez la section Créer une table dans DynamoDB dans le manuel Amazon DynamoDB Developer Guide.

Utilisez ce qui suit pour les détails du tableau :

- Dans Nom de la table, entrez le nom de table de votre choix.
- Pour la clé de partition, entrez votre propre ID d'instance.
- Vous pouvez utiliser les paramètres par défaut pour les paramètres du tableau

AWS CLI

Exécutez la commande suivante pour créer une table DynamoDB.

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

Étape 7 : Création d'une AWS Lambda fonction (console)

La fonction Lambda sert de moteur de traitement principal, remplaçant la logique d'évaluation du modèle de détecteur de. AWS IoT Events Dans cet exemple, nous nous intégrons à d'autres AWS services pour gérer les données entrantes, gérer l'état et déclencher des actions en fonction des règles que vous avez définies.

Créez une fonction Lambda avec NodeJS runtime. Utilisez l'extrait de code suivant pour remplacer les constantes codées en dur :

1. Ouvrez la AWS Lambda console.
2. Choisissez Créer une fonction.
3. Entrez un nom pour le nom de la fonction.
4. Sélectionnez NodeJS 22.x comme environnement d'exécution.
5. Dans la liste déroulante Modifier le rôle d'exécution par défaut, choisissez Utiliser le rôle existant, puis sélectionnez le rôle IAM que vous avez créé lors des étapes précédentes.

6. Choisissez Créer une fonction.
7. Collez l'extrait de code suivant après avoir remplacé les constantes codées en dur.
8. Une fois votre fonction créée, sous l'onglet Code, collez l'exemple de code suivant, en remplaçant le **`your-destination-endpoint`** point de terminaison par le vôtre.

```
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
        ]
    );
```

Étape 8 : ajouter un déclencheur Amazon Kinesis Data Streams

Ajoutez un déclencheur Kinesis Data Streams à la fonction Lambda à l'aide du ou. AWS Management Console AWS CLI

L'ajout d'un déclencheur Kinesis Data Streams à votre fonction Lambda établit le lien entre votre pipeline d'ingestion de données et votre logique de traitement, ce qui lui permet d'évaluer automatiquement les flux de données IoT entrants et de réagir aux événements en temps réel, de la même manière que le traitement des entrées. AWS IoT Events

Console

Pour plus d'informations, consultez la section Créer un mappage de source d'événements pour appeler une fonction Lambda dans le Guide du AWS Lambda développeur.

Utilisez ce qui suit pour les détails du mappage des sources d'événements :

- Pour Nom de la fonction, entrez le nom lambda utilisé dansÉtape 7 : Création d'une AWS Lambda fonction (console).
- Pour Consumer : facultatif, entrez l'ARN de votre flux Kinesis.
- Pour la Taille de lot, saisissez **10**.

AWS CLI

Exécutez la commande suivante pour créer le déclencheur de la fonction Lambda.

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

Étape 9 : Tester les fonctionnalités d'ingestion et de sortie des données (AWS CLI)

Publiez une charge utile dans la rubrique MQTT en fonction de ce que vous avez défini dans votre modèle de détecteur. Voici un exemple de charge utile pour la rubrique MQTT `your-topic-name` pour tester une implémentation.

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

Vous devriez voir un message MQTT publié sur un sujet avec le contenu suivant (ou similaire) :

```
{
```

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

Procédure de migration pour les AWS IoT SiteWise alarmes dans AWS IoT Events

Cette section décrit les solutions alternatives qui fournissent des fonctionnalités d'alarme similaires à celles que vous utilisez lors de la migration AWS IoT Events.

Pour les AWS IoT SiteWise propriétés qui utilisent des AWS IoT Events alarmes, vous pouvez migrer vers une solution utilisant des CloudWatch alarmes. Cette approche fournit des capacités de surveillance robustes avec des fonctionnalités établies SLAs et supplémentaires telles que la détection des anomalies et les alarmes groupées.

Comparaison des architectures

La configuration AWS IoT Events d'alarme actuelle pour les AWS IoT SiteWise propriétés doit être créée `AssetModelCompositeModels` dans le modèle d'actif, comme décrit dans la section Définition des alarmes externes du guide de AWS IoT SiteWise l'utilisateur. AWS IoT SiteWise Les modifications apportées à la nouvelle solution sont généralement gérées via la AWS IoT Events console.

La nouvelle solution permet de gérer les alarmes en tirant parti des CloudWatch alarmes. Cette approche utilise des AWS IoT SiteWise notifications pour publier des points de données de propriété dans des rubriques AWS IoT Core MQTT, qui sont ensuite traités par une fonction Lambda. La fonction transforme ces notifications en CloudWatch métriques, permettant de surveiller les alarmes grâce à un cadre CloudWatch d'alarme.

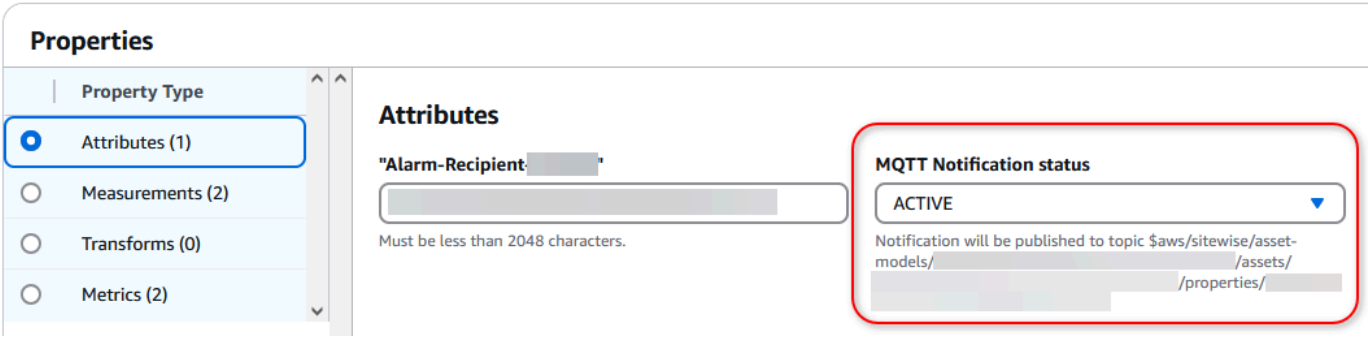
Objectif	Solution	Différences
Source de données : données de propriété provenant de AWS IoT SiteWise	AWS IoT SiteWise Notifications MQTT	Remplace l'intégration directe d'IoT Events par les notifications MQTT provenant des propriété s AWS IoT SiteWise
Traitement des données — Transform	Fonction Lambda	Traite les notifications relatives aux AWS IoT SiteWise propriétés et les convertit en CloudWatch métriques

Objectif	Solution	Différences
e les données de propriété		
Évaluation des alarmes — Surveille les métriques et déclenche des alarmes	CloudWatch Alarmes Amazon	Remplace AWS IoT Events les alarmes par des CloudWatch alarmes, offrant des fonctionn alités supplémentaires telles que la détection des anomalies
Intégration — Connexion avec AWS IoT SiteWise	AWS IoT SiteWise alarmes externes	Possibilité optionnelle de réimporter CloudWatc h les alarmes en AWS IoT SiteWise tant qu'alarmes externes

Étape 1 : activer les notifications MQTT sur la propriété de l'actif

Si vous utilisez AWS IoT Events des intégrations pour les AWS IoT SiteWise alarmes, vous pouvez activer les notifications MQTT pour chaque propriété à surveiller.

1. Suivez la AWS IoT SiteWise procédure de configuration des alarmes sur les actifs jusqu'à ce que vous passiez à chaque étape de modification des propriétés du modèle d'actifs.
2. Pour chaque propriété à migrer, remplacez le statut de notification MQTT par ACTIVE.



3. Notez le chemin de rubrique vers lequel l'alarme est publiée pour chaque attribut d'alarme modifié.

Pour plus d'informations, consultez les ressources de documentation suivantes :

- Découvrez les propriétés des actifs dans les rubriques MQTT du guide de l'AWS IoT SiteWise utilisateur.
- Rubriques relatives au MQTT dans le guide du AWS IoT développeur.

Étape 2 : Création d'une AWS Lambda fonction

Créez une fonction Lambda pour lire le tableau TQV publié par le sujet MQTT et publiez des valeurs individuelles dans. CloudWatch Nous utiliserons cette fonction Lambda comme action de destination à déclencher dans AWS IoT Core Message Rules.

1. Ouvrez la AWS Lambda console.
2. Choisissez Créer une fonction.
3. Entrez un nom pour le nom de la fonction.
4. Sélectionnez NodeJS 22.x comme environnement d'exécution.
5. Dans la liste déroulante Modifier le rôle d'exécution par défaut, choisissez Utiliser le rôle existant, puis sélectionnez le rôle IAM que vous avez créé lors des étapes précédentes.

   > **Note**
   >
   > Cette procédure suppose que vous avez déjà migré votre modèle de détecteur. Si vous n'avez pas de rôle IAM, consultez???.

6. Choisissez Créer une fonction.
7. Collez l'extrait de code suivant après avoir remplacé les constantes codées en dur.

```
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)}')
        }
```

Étape 3 : Création d'une règle de routage des AWS IoT Core messages

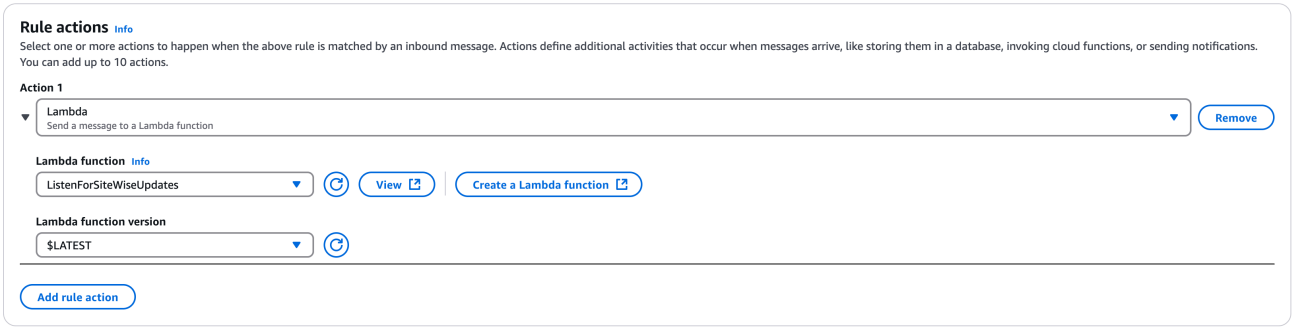
- Suivez le didacticiel : procédure de republication d'un message MQTT en saisissant les informations suivantes lorsque vous y êtes invité :

  a. Nom de la règle de routage des messages`SiteWiseToCloudwatchAlarms`.

  b. Pour la requête, vous pouvez utiliser ce qui suit :

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

  c. Dans Actions de règles, sélectionnez l'action Lambda à partir de laquelle envoyer les données générées à AWS IoT SiteWise . CloudWatch Par exemple :



Étape 4 : Afficher les CloudWatch métriques

Lorsque vous ingérez des données AWS IoT SiteWise, la propriété sélectionnée précédemment achemine les données vers la fonction Lambda que nous avons créée dansÉtape 1 : activer les notifications MQTT sur la propriété de l'actif. Étape 2 : Création d'une AWS Lambda fonction Au cours de cette étape, vous pouvez vérifier que le Lambda envoie vos métriques à. CloudWatch

1. Ouvrez la CloudWatch AWS Management Console.
2. Dans le volet de navigation de gauche, choisissez Metrics, puis All metrics.
3. Choisissez l'URL d'une alarme pour l'ouvrir.
4. Sous l'onglet Source, le CloudWatch résultat est similaire à celui de cet exemple. Ces informations de source confirment que les données métriques alimentent CloudWatch.

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

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

Étape 5 : créer des CloudWatch alarmes

Suivez la procédure de création CloudWatch d'une alarme basée sur un seuil statique du guide de CloudWatch l'utilisateur Amazon pour créer des alarmes pour chaque métrique pertinente.

> **Note**
>
> Il existe de nombreuses options de configuration des alarmes dans Amazon. CloudWatch Pour plus d'informations sur les CloudWatch alarmes, consultez la section Utilisation des CloudWatch alarmes Amazon dans le guide de CloudWatch l'utilisateur Amazon.

Étape 6 : (Facultatif) importez l' CloudWatch alarme dans AWS IoT SiteWise

Vous pouvez configurer des CloudWatch alarmes pour qu'elles renvoient des données à AWS IoT SiteWise l'aide CloudWatch d'actions d'alarme et de Lambda. Cette intégration vous permet de visualiser l'état et les propriétés des alarmes dans le portail SiteWise Monitor.

1. Configurez l'alarme externe en tant que propriété dans un modèle d'actif. Pour plus d'informations, reportez-vous à la section Définition des alarmes externes AWS IoT SiteWise dans le guide de AWS IoT SiteWise l'utilisateur.
2. Créez une fonction Lambda qui utilise l'BatchPutAssetPropertyValueAPI figurant dans le guide de l'AWS IoT SiteWise utilisateur pour envoyer des données d'alarme à. AWS IoT SiteWise
3. Configurez des actions CloudWatch d'alarme pour appeler votre fonction Lambda lorsque l'état de l'alarme change. Pour plus d'informations, consultez la section Actions d'alarme dans le guide de CloudWatch l'utilisateur Amazon.

Con AWS IoT Events figuration

Cette section fournit un guide de configuration AWS IoT Events, y compris la création d'un AWS compte, la configuration des autorisations nécessaires et la définition des rôles pour gérer l'accès aux ressources.

Rubriques

- Configuration d'un Compte AWS
- Configuration des autorisations pour AWS IoT Events

Configuration d'un Compte AWS

Inscrivez-vous pour un Compte AWS

Si vous n'en avez pas Compte AWS, procédez comme suit pour en créer un.

Pour vous inscrire à un Compte AWS

1. Ouvrez l'https://portal.aws.amazon.com/billing/inscription.
2. Suivez les instructions en ligne.

   Dans le cadre de la procédure d’inscription, vous recevrez un appel téléphonique ou un SMS et vous saisirez un code de vérification en utilisant le clavier numérique du téléphone.

   Lorsque vous vous inscrivez à un Compte AWS, un Utilisateur racine d'un compte AWSest créé. Par défaut, seul l’utilisateur racine a accès à l’ensemble des Services AWS et des ressources de ce compte. La meilleure pratique de sécurité consiste à attribuer un accès administratif à un utilisateur, et à utiliser uniquement l’utilisateur racine pour effectuer les tâches nécessitant un accès utilisateur racine.

AWS vous envoie un e-mail de confirmation une fois le processus d'inscription terminé. À tout moment, vous pouvez consulter l'activité actuelle de votre compte et gérer votre compte en accédant à https://aws.amazon.com/et en choisissant Mon compte.

Création d’un utilisateur doté d’un accès administratif

Après vous être inscrit à un Compte AWS, sécurisez Utilisateur racine d'un compte AWS AWS IAM Identity Center, activez et créez un utilisateur administratif afin de ne pas utiliser l'utilisateur root pour les tâches quotidiennes.

Sécurisez votre Utilisateur racine d'un compte AWS

1. Connectez-vous en AWS Management Consoletant que propriétaire du compte en choisissant Utilisateur root et en saisissant votre adresse Compte AWS e-mail. Sur la page suivante, saisissez votre mot de passe.

   Pour obtenir de l’aide pour vous connecter en utilisant l’utilisateur racine, consultez Connexion en tant qu’utilisateur racine dans le Guide de l’utilisateur Connexion à AWS .
2. Activez l’authentification multifactorielle (MFA) pour votre utilisateur racine.

   Pour obtenir des instructions, voir Activer un périphérique MFA virtuel pour votre utilisateur Compte AWS root (console) dans le guide de l'utilisateur IAM.

Création d’un utilisateur doté d’un accès administratif

1. Activez IAM Identity Center.

   Pour obtenir des instructions, consultez Activation d’ AWS IAM Identity Center dans le Guide de l’utilisateur AWS IAM Identity Center .
2. Dans IAM Identity Center, octroyez un accès administratif à un utilisateur.

   Pour un didacticiel sur l'utilisation du Répertoire IAM Identity Center comme source d'identité, voir Configurer l'accès utilisateur par défaut Répertoire IAM Identity Center dans le Guide de AWS IAM Identity Center l'utilisateur.

Connexion en tant qu’utilisateur doté d’un accès administratif

- Pour vous connecter avec votre utilisateur IAM Identity Center, utilisez l’URL de connexion qui a été envoyée à votre adresse e-mail lorsque vous avez créé l’utilisateur IAM Identity Center.

  Pour obtenir de l'aide pour vous connecter en utilisant un utilisateur d'IAM Identity Center, consultez la section Connexion au portail AWS d'accès dans le guide de l'Connexion à AWS utilisateur.

Attribution d'un accès à d'autres utilisateurs

1. Dans IAM Identity Center, créez un ensemble d'autorisations qui respecte la bonne pratique consistant à appliquer les autorisations de moindre privilège.

   Pour obtenir des instructions, consultez Création d'un ensemble d'autorisations dans le Guide de l'utilisateur AWS IAM Identity Center .

2. Attribuez des utilisateurs à un groupe, puis attribuez un accès par authentification unique au groupe.

   Pour obtenir des instructions, consultez Ajout de groupes dans le Guide de l'utilisateur AWS IAM Identity Center .

Configuration des autorisations pour AWS IoT Events

La mise en œuvre d'autorisations appropriées est importante pour une utilisation sûre et efficace de AWS IoT Events. Cette section décrit les autorisations requises pour utiliser certaines fonctionnalités de AWS IoT Events. Vous pouvez utiliser des AWS CLI commandes ou la console Gestion des identités et des accès AWS (IAM) pour créer des rôles et des politiques d'autorisation associées afin d'accéder aux ressources ou d'exécuter certaines fonctions dans AWS IoT Events.

Le guide de l'utilisateur IAM contient des informations plus détaillées sur le contrôle sécurisé des autorisations d'accès aux AWS ressources. Pour des informations spécifiques à AWS IoT Events, voir Actions, ressources et clés de condition pour AWS IoT Events.

Pour utiliser la console IAM pour créer et gérer des rôles et des autorisations, consultez le didacticiel IAM : déléguer l'accès entre AWS comptes à l'aide de rôles IAM.

> **Note**
>
> Les clés peuvent comporter de 1 à 128 caractères et peuvent inclure :
>
> - lettres majuscules ou minuscules de A à Z
> - chiffres de 0 à 9
> - caractères spéciaux -, _ ou :.

Autorisations d'action pour AWS IoT Events

AWS IoT Events vous permet de déclencher des actions utilisant d'autres AWS services. Pour ce faire, vous devez AWS IoT Events autoriser l'exécution de ces actions en votre nom. Cette section contient une liste des actions et un exemple de politique qui autorise l'exécution de toutes ces actions sur vos ressources. Modifiez les *account-id* références *region* et selon les besoins. Dans la mesure du possible, vous devez également modifier les caractères génériques (*) pour faire référence aux ressources spécifiques auxquelles vous aurez accès. Vous pouvez utiliser la console IAM pour autoriser l'envoi AWS IoT Events d'une alerte Amazon SNS que vous avez définie.

AWS IoT Events prend en charge les actions suivantes qui vous permettent d'utiliser un temporisateur ou de définir une variable :

- `setTimer`pour créer un chronomètre.
- `resetTimer`pour réinitialiser le chronomètre.
- `clearTimer`pour supprimer le chronomètre.
- `setVariable`pour créer une variable.

AWS IoT Events prend en charge les actions suivantes qui vous permettent de travailler avec AWS les services :

- `iotTopicPublish`pour publier un message sur un sujet MQTT.
- `iotEvents`pour envoyer des données en AWS IoT Events tant que valeur d'entrée.
- `iotSiteWise` pour envoyer des données à une propriété de ressources dans AWS IoT SiteWise.
- `dynamoDB`pour envoyer des données vers une table Amazon DynamoDB.
- `dynamoDBv2`pour envoyer des données vers une table Amazon DynamoDB.
- `firehose`pour envoyer des données vers un flux Amazon Data Firehose.
- `lambda`pour invoquer une AWS Lambda fonction.
- `sns`pour envoyer des données sous forme de notification push.
- `sqs`pour envoyer des données vers une file d'attente Amazon SQS.

Example Politique

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:*"
        }
    ]
}
```

Sécurisation des données d'entrée dans AWS IoT Events

Il est important de déterminer qui peut autoriser l'accès aux données d'entrée destinées à être utilisées dans un modèle de détecteur. Si vous souhaitez restreindre les autorisations globales d'un utilisateur ou d'une entité, mais qui est autorisé à créer ou à mettre à jour un modèle de détecteur, vous devez également autoriser cet utilisateur ou cette entité à mettre à jour le routage des entrées. Cela signifie qu'en plus d'accorder une autorisation pour `iotevents:CreateDetectorModel` et`iotevents:UpdateDetectorModel`, vous devez également accorder une autorisation pour`iotevents:UpdateInputRouting`.

Example

La politique suivante ajoute l'autorisation pour`iotevents:UpdateInputRouting`.

JSON

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

Vous pouvez spécifier une liste d'entrées Amazon Resource Names (ARNs) au lieu du caractère générique « * » pour le « `Resource` » afin de limiter cette autorisation à des entrées spécifiques. Cela vous permet de restreindre l'accès aux données d'entrée consommées par les modèles de détecteurs créés ou mis à jour par l'utilisateur ou l'entité.

Politique relative aux rôles de CloudWatch journalisation d'Amazon pour AWS IoT Events

Les documents de politique suivants fournissent la politique de rôle et la politique de confiance qui AWS IoT Events permettent de soumettre des journaux CloudWatch en votre nom.

Politique de rôle :

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

Stratégie d'approbation :

JSON

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

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

Vous avez également besoin d'une politique d'autorisation IAM attachée à l'utilisateur qui permet à celui-ci de transmettre des rôles, comme suit. Pour plus d'informations, consultez la section Accorder à un utilisateur l'autorisation de transmettre un rôle à un AWS service dans le Guide de l'utilisateur IAM.

JSON

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

Vous pouvez utiliser la commande suivante pour définir la politique de ressources pour les CloudWatch journaux. Cela permet AWS IoT Events de placer les événements du journal dans CloudWatch des flux.

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

Utilisez la commande suivante pour définir les options de journalisation. Remplacez le `roleArn` par le rôle de journalisation que vous avez créé.

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

Politique relative aux rôles de messagerie Amazon SNS pour AWS IoT Events

L'intégration AWS IoT Events à Amazon SNS nécessite une gestion minutieuse des autorisations afin de garantir la sécurité et l'efficacité de l'envoi des notifications. Ce guide explique le processus de configuration des rôles et des politiques IAM pour autoriser la publication AWS IoT Events de messages sur les rubriques Amazon SNS.

Les documents de politique suivants fournissent la politique de rôle et la politique de confiance qui permettent AWS IoT Events d'envoyer des messages SNS.

Politique de rôle :

JSON

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

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

Stratégie d'approbation :

JSON

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

Commencer à utiliser la AWS IoT Events console

Cette section explique comment créer une entrée et un modèle de détecteur à l'aide de la AWS IoT Events console. Vous modélisez deux états d'un moteur : un état normal et un état de surpression. Lorsque la pression mesurée dans le moteur dépasse un certain seuil, le modèle passe de l'état normal à l'état de surpression. Il envoie ensuite un message Amazon SNS pour avertir un technicien du problème. Lorsque la pression chute à nouveau en dessous du seuil pour trois relevés de pression consécutifs, le modèle revient à l'état normal et envoie un autre message Amazon SNS en guise de confirmation.

Nous vérifions trois relevés consécutifs en dessous du seuil de pression afin d'éliminer tout bégaiement éventuel dû à une surpression ou à des messages normaux, en cas de phase de rétablissement non linéaire ou de lecture de pression anormale.

Sur la console, vous pouvez également trouver plusieurs modèles de détecteurs prédéfinis que vous pouvez personnaliser. Vous pouvez également utiliser la console pour importer des modèles de détecteurs que d'autres ont écrits ou pour exporter vos modèles de détecteurs et les utiliser dans différentes AWS régions. Si vous importez un modèle de détecteur, assurez-vous de créer les entrées requises ou de les recréer pour la nouvelle région, et de mettre à jour les rôles ARNs utilisés.

Utilisez la AWS IoT Events console pour en savoir plus sur les points suivants.

Définir les entrées

Pour surveiller vos appareils et processus, vous devez pouvoir obtenir des données télémétriques dans AWS IoT Events. Cela se fait en envoyant des messages en tant qu'entrées à AWS IoT Events. Pour ce faire, plusieurs options s'offrent à vous :

- Utilisez l' BatchPutMessageopération.
- Dans AWS IoT Core, écrivez une règle AWS IoT Events d'action pour le moteur de AWS IoT règles qui transfère les données de vos messages vers AWS IoT Events. Vous devez identifier l'entrée par son nom.
- Dans AWS IoT Analytics, utilisez l' CreateDatasetopération pour créer un ensemble de données avec`contentDeliveryRules`. Ces règles spécifient l' AWS IoT Events entrée à laquelle le contenu de l'ensemble de données est envoyé automatiquement.

Avant que vos appareils puissent envoyer des données de cette manière, vous devez définir une ou plusieurs entrées. Pour ce faire, attribuez un nom à chaque entrée et spécifiez les champs des données des messages entrants surveillés par l'entrée.

Création d'un modèle de détecteur

Définissez un modèle de détecteur (un modèle de votre équipement ou de votre processus) à l'aide des états. Pour chaque état, définissez une logique conditionnelle (booléenne) qui évalue les entrées entrantes afin de détecter les événements significatifs. Lorsque le modèle de détecteur détecte un événement, il peut modifier l'état ou lancer des actions personnalisées ou prédéfinies à l'aide d'autres AWS services. Vous pouvez définir des événements supplémentaires qui déclenchent des actions lors de l'entrée ou de la sortie d'un état et, éventuellement, lorsqu'une condition est remplie.

Dans ce didacticiel, vous envoyez un message Amazon SNS en tant qu'action lorsque le modèle entre ou sort d'un certain état.

Surveiller un appareil ou un processus

Si vous surveillez plusieurs appareils ou processus, spécifiez un champ dans chaque entrée qui identifie le périphérique ou le processus d'où provient l'entrée. Voir le `key` champ dans`CreateDetectorModel`. Lorsque le champ de saisie identifié par le `key` reconnaît une nouvelle valeur, un nouveau dispositif est identifié et un détecteur est créé. Chaque détecteur est une instance du modèle de détecteur. Le nouveau détecteur continue de répondre aux entrées provenant de cet appareil jusqu'à ce que son modèle de détecteur soit mis à jour ou supprimé.

Si vous surveillez un seul processus (même si plusieurs appareils ou sous-processus envoient des entrées), vous ne spécifiez pas de `key` champ d'identification unique. Dans ce cas, le modèle crée un seul détecteur (instance) lorsque la première entrée arrive.

Envoyez des messages sous forme d'entrées à votre modèle de détecteur

Il existe plusieurs manières d'envoyer un message à partir d'un appareil ou de le traiter en tant qu'entrée dans un AWS IoT Events détecteur sans que vous ayez à appliquer un formatage supplémentaire au message. Dans ce didacticiel, vous allez utiliser la AWS IoT console pour écrire une règle AWS IoT Events d'action pour le moteur de AWS IoT règles qui transfère les données de vos messages AWS IoT Events.

Pour ce faire, identifiez l'entrée par son nom et continuez à utiliser la AWS IoT console pour générer des messages qui sont transférés en tant qu'entrées à AWS IoT Events.

> **Note**
>
> Ce didacticiel utilise la console pour créer la même choseinput, `detector model` comme indiqué dans l'exemple surTutoriels pour les cas d' AWS IoT Events utilisation. Vous pouvez utiliser cet exemple JSON pour vous aider à suivre le didacticiel.

Rubriques

- Prérequis pour commencer AWS IoT Events
- Créez une entrée pour les modèles dans AWS IoT Events
- Créez un modèle de détecteur dans AWS IoT Events
- Envoyez des entrées pour tester le modèle de détecteur dans AWS IoT Events

Prérequis pour commencer AWS IoT Events

Si vous n'avez pas de AWS compte, créez-en un.

1. Suivez les étapes ci-dessous Con AWS IoT Events figuration pour vous assurer que la configuration du compte et les autorisations sont correctes.
2. Créez deux rubriques Amazon Simple Notification Service (Amazon SNS).

   Ce didacticiel (et l'exemple correspondant) supposent que vous avez créé deux rubriques Amazon SNS. Ces rubriques sont présentées sous la forme : `arn:aws:sns:us-east-1:123456789012:underPressureAction` et`arn:aws:sns:us-east-1:123456789012:pressureClearedAction`. ARNs Remplacez ces valeurs par ARNs les rubriques Amazon SNS que vous créez. Pour de plus amples informations, consultez dans le Guide du développeur Amazon Simple Notification Service.

   Au lieu de publier des alertes sur des sujets Amazon SNS, vous pouvez demander aux détecteurs d'envoyer des messages MQTT avec un sujet que vous spécifiez. Avec cette option, vous pouvez vérifier que votre modèle de détecteur crée des instances et que ces instances envoient des alertes en utilisant la console AWS IoT Core pour vous abonner et surveiller les messages envoyés à ces sujets MQTT. Vous pouvez également définir le nom du sujet MQTT de manière dynamique lors de l'exécution à l'aide d'une entrée ou d'une variable créée dans le modèle de détecteur.

3. Choisissez un modèle Région AWS qui soutient AWS IoT Events. Pour plus d'informations, consultez AWS IoT Events dans le Références générales AWS. Pour obtenir de l'aide, consultez la section Prise en main d'un service AWS Management Console dans la section Mise en route avec le AWS Management Console.

Créez une entrée pour les modèles dans AWS IoT Events

Lorsque vous créez les entrées pour vos modèles, nous vous recommandons de collecter des fichiers contenant des exemples de charges utiles de messages que vos appareils ou processus envoient pour signaler leur état de santé. Le fait de disposer de ces fichiers vous permet de définir les entrées requises.

Vous pouvez créer une entrée à l'aide de plusieurs méthodes décrites dans cette section.

Création d'un fichier d'entrée JSON

1. Pour commencer, créez un fichier nommé `input.json` sur votre système de fichiers local avec le contenu suivant :

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

2. Maintenant que vous avez ce `input.json` fichier de démarrage, vous pouvez créer une entrée. Il existe deux manières de créer une entrée. Vous pouvez créer une entrée à l'aide du volet de navigation de la AWS IoT Events console. Vous pouvez également créer une entrée dans le modèle de détecteur une fois celui-ci créé.

Création et configuration d'une entrée

Apprenez à créer une entrée, pour un modèle d'alarme ou un modèle de détecteur.

1. Connectez-vous à la AWS IoT Events console ou sélectionnez l'option Créer un nouveau AWS IoT Events compte.

2. Dans le coin supérieur gauche de la AWS IoT Events console, sélectionnez et développez le volet de navigation.
3. Dans le volet de navigation de gauche, sélectionnez Entrées.
4. Dans le coin droit de la console, choisissez Create input.
5. Fournissez un objet unique InputName.
6. Facultatif : entrez une description pour votre saisie.
7. Pour télécharger un fichier JSON, sélectionnez le `input.json` fichier que vous avez créé dans l'aperçu pourCréation d'un fichier d'entrée JSON. L'option Choisir les attributs d'entrée s'affiche avec la liste des attributs que vous avez saisis.
8. Pour Choisir les attributs d'entrée, sélectionnez les attributs à utiliser, puis choisissez Créer. Dans cet exemple, nous sélectionnons motorid et SensorData.pressure.
9. Facultatif — Ajoutez des balises pertinentes à l'entrée.

> **Note**
>
> Vous pouvez également créer des entrées supplémentaires dans le modèle de détecteur dans la AWS IoT Events console. Pour de plus amples informations, veuillez consulter Créez une entrée dans le modèle de détecteur dans AWS IoT Events.

Créez une entrée dans le modèle de détecteur dans AWS IoT Events

Les entrées du détecteur AWS IoT Events servent de pont entre vos sources de données et les modèles de détecteurs. Les entrées du détecteur fournissent les données brutes qui alimentent les capacités de détection d'événements et d'automatisation de AWS IoT Events. Apprenez à configurer les entrées des détecteurs pour aider vos modèles à répondre avec précision aux événements et aux conditions du monde réel dans votre écosystème IoT.

Cette section explique comment définir une entrée pour qu'un modèle de détecteur reçoive des données ou des messages de télémétrie.

Pour définir une entrée pour un modèle de détecteur

1. Ouvrez la AWS IoT Events console.
2. Dans la AWS IoT Events console, choisissez Créer un modèle de détecteur.
3. Choisissez Créer.

4. Choisissez Créer une entrée.
5. Pour la saisie, entrez une InputNamedescription facultative, puis choisissez Charger le fichier. Dans la boîte de dialogue qui s'affiche, sélectionnez le `input.json` fichier que vous avez créé dans l'aperçu pourCréation d'un fichier d'entrée JSON.
6. Pour Choisir les attributs d'entrée, sélectionnez les attributs à utiliser, puis choisissez Créer. Dans cet exemple, nous sélectionnons MotorID et SensorData.pressure.

Créez un modèle de détecteur dans AWS IoT Events

Dans cette rubrique, vous allez définir un modèle de détecteur (un modèle de votre équipement ou de votre processus) à l'aide des états.

Pour chaque état, vous définissez une logique conditionnelle (booléenne) qui évalue les entrées entrantes afin de détecter un événement significatif. Lorsqu'un événement est détecté, il change d'état et peut lancer des actions supplémentaires. Ces événements sont appelés événements de transition.

Dans vos états, vous définissez également des événements qui peuvent exécuter des actions chaque fois que le détecteur entre ou sort de cet état ou lorsqu'une entrée est reçue (ces événements sont appelés `OnEnter` `OnInput` événements `OnExit` et). Les actions ne sont exécutées que si la logique conditionnelle de l'événement est évaluée à`true`.

Pour créer un modèle de détecteur

1. Le premier état du détecteur a été créé pour vous. Pour le modifier, sélectionnez le cercle avec le libellé State_1 dans l'espace d'édition principal.
2. Dans le volet État, entrez le nom de l'État et OnEnterchoisissez Ajouter un événement.
3. Sur la page Ajouter un OnEnter événement, entrez le nom de l'événement et la condition de l'événement. Dans cet exemple, entrez `true` pour indiquer que l'événement est toujours initié lorsque l'état est saisi.
4. Sous Actions liées aux événements, choisissez Ajouter une action.
5. Sous Actions liées aux événements, procédez comme suit :

    a. Sélectionnez Définir une variable

    b. Pour Fonctionnement variable, choisissez Affecter une valeur.

    c. Dans Nom de la variable, entrez le nom de la variable à définir.

d. Pour Valeur variable, entrez la valeur **`0`** (zéro).

6. Choisissez Enregistrer.

   Une variable, comme celle que vous avez définie, peut être définie (avec une valeur) dans tous les cas dans le modèle de détecteur. La valeur de la variable ne peut être référencée (par exemple, dans la logique conditionnelle d'un événement) qu'une fois que le détecteur a atteint un état et a exécuté une action dans laquelle elle est définie ou définie.

7. Dans le volet État, cliquez sur le X à côté de State pour revenir à la palette du modèle de détecteur.
8. Pour créer un deuxième état de détecteur, dans la palette du modèle de détecteur, choisissez State et faites-le glisser dans l'espace d'édition principal. Cela crée un état intituléuntitled_state_1.
9. Faites une pause sur le premier état (Normal). Une flèche apparaît sur le pourtour de l'État.
10. Cliquez et faites glisser la flèche du premier état au second état. Une ligne dirigée entre le premier état et le second état (intitulée Sans titre) apparaît.
11. Sélectionnez la ligne Sans titre. Dans le volet des événements de transition, entrez un nom d'événement et une logique de déclenchement d'événement.
12. Dans le volet des événements de transition, choisissez Ajouter une action.
13. Dans le volet Ajouter des actions relatives à un événement de transition, choisissez Ajouter une action.
14. Pour Choisir une action, choisissez Définir une variable.

    a. Pour Fonctionnement variable, choisissez Affecter une valeur.

    b. Dans Nom de la variable, entrez le nom de la variable.

    c. Pour Attribuer une valeur, entrez une valeur telle que : `$variable.pressureThresholdBreached + 3`

    d. Choisissez Enregistrer.

15. Sélectionnez le deuxième état untitled_state_1.
16. Dans le volet État, entrez le nom de l'État et pour En entrée, choisissez Ajouter un événement.
17. Sur la page Ajouter OnEnter un événement, entrez le nom de l'événement et la condition de l'événement. Choisissez Add action.
18. Pour Choisir une action, choisissez Envoyer un message SNS.

    a. Pour la rubrique SNS, entrez l'ARN cible de votre rubrique Amazon SNS.

b. Choisissez Enregistrer.

19. Continuez à ajouter les événements dans l'exemple.

a. Pour OnInput, choisissez Ajouter un événement, puis entrez et enregistrez les informations d'événement suivantes.

```
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. Pour OnInput, choisissez Ajouter un événement, puis entrez et enregistrez les informations d'événement suivantes.

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

c. Pour OnExit, choisissez Ajouter un événement, puis entrez et enregistrez les informations d'événement suivantes à l'aide de l'ARN de la rubrique Amazon SNS que vous avez créée.

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

20. Pause sur le deuxième état (Dangereux). Une flèche apparaît sur le pourtour de l'État

21. Cliquez et faites glisser la flèche du deuxième état au premier état. Une ligne dirigée avec le libellé Sans titre apparaît.

22. Choisissez la ligne Sans titre et dans le volet des événements de transition, entrez un nom d'événement et une logique de déclenchement d'événement à l'aide des informations suivantes.

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

Pour plus d'informations sur les raisons pour lesquelles nous testons la `$input` valeur et la `$variable` valeur dans la logique de déclenchement, consultez l'entrée relative à la disponibilité des valeurs des variables dansAWS IoT Events restrictions et limites du modèle de détecteur.

23. Sélectionnez l'état de départ. Par défaut, cet état a été créé lorsque vous avez créé un modèle de détecteur). Dans le volet Démarrer, choisissez l'état de destination (par exemple, Normal).
24. Configurez ensuite le modèle de votre détecteur pour qu'il écoute les entrées. Dans le coin supérieur droit, choisissez Publier.
25. Sur la page du modèle de détecteur de publication, procédez comme suit.

    a. Entrez un nom de modèle de détecteur, une description et le nom d'un rôle. Ce rôle a été créé pour vous.

    b. Choisissez Créer un détecteur pour chaque valeur de clé unique. Pour créer et utiliser votre propre rôle, suivez les étapes décrites Configuration des autorisations pour AWS IoT Events et saisissez-le comme rôle ici.
26. Pour la clé de création du détecteur, choisissez le nom de l'un des attributs de l'entrée que vous avez définie précédemment. L'attribut que vous choisissez comme clé de création du détecteur doit être présent dans chaque message saisi et doit être unique pour chaque appareil qui envoie des messages. Cet exemple utilise l'attribut motorid.
27. Choisissez Enregistrer et publier.

> **Note**
>
> Le nombre de détecteurs uniques créés pour un modèle de détecteur donné est basé sur les messages d'entrée envoyés. Lorsqu'un modèle de détecteur est créé, une clé est sélectionnée parmi les attributs d'entrée. Cette clé détermine l'instance de détecteur à utiliser. Si la clé n'a jamais été vue auparavant (pour ce modèle de détecteur), une nouvelle

instance de détecteur est créée. Si la clé a déjà été vue, nous utilisons l'instance de détecteur existante correspondant à cette valeur de clé.

Vous pouvez créer une copie de sauvegarde de la définition de votre modèle de détecteur (au format JSON), recréer ou mettre à jour le modèle de détecteur ou l'utiliser comme modèle pour créer un autre modèle de détecteur.

Vous pouvez le faire depuis la console ou à l'aide de la commande CLI suivante. Si nécessaire, modifiez le nom du modèle de détecteur pour qu'il corresponde à celui que vous avez utilisé lors de sa publication à l'étape précédente.

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

Cela crée un fichier (`motorDetectorModel.json`) dont le contenu est similaire au suivant.

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

Envoyez des entrées pour tester le modèle de détecteur dans AWS IoT Events

Il existe plusieurs manières de recevoir des données de télémétrie dans AWS IoT Events (voirActions prises en charge pour recevoir des données et déclencher des actions dans AWS IoT Events). Cette rubrique explique comment créer une AWS IoT règle dans la AWS IoT console qui transmet les messages sous forme d'entrées à votre AWS IoT Events détecteur. Vous pouvez utiliser le client MQTT de la AWS IoT console pour envoyer des messages de test. Vous pouvez utiliser cette méthode pour obtenir des données de télémétrie AWS IoT Events lorsque vos appareils sont en mesure d'envoyer des messages MQTT à l'aide du AWS IoT courtier de messages.

Pour envoyer des entrées afin de tester le modèle de détecteur

1. Ouvrez la AWS IoT Core console. Dans le volet de navigation de gauche, sous Gérer, choisissez Routage des messages, puis Règles.
2. Choisissez Créer une règle dans le coin supérieur droit.
3. Sur la page Créer une règle, procédez comme suit :

   1. Étape 1. Spécifiez les propriétés des règles. Renseignez les champs suivants :
      - Nom de la règle. Entrez un nom pour votre règle, par exemple`MyIoTEventsRule`.

        > **Note**
        >
        > N'utilisez pas d'espaces.

      - Description de la règle Ce nom est facultatif.
      - Choisissez Suivant.
   2. Étape 2. Configurez l'instruction SQL. Renseignez les champs suivants :
      - Version SQL. Sélectionnez l'option appropriée dans la liste.
      - instruction SQL. Saisissez **`SELECT *, topic(2) as motorid FROM 'motors/+/status'`**.

      Choisissez Suivant.
   3. Étape 3. Attachez des actions aux règles. Dans la section Actions relatives aux règles, effectuez les opérations suivantes :
      - Mesure 1. Sélectionnez IoT Events. Les champs suivants apparaissent :
        a. Nom d'entrée. Sélectionnez l'option appropriée dans la liste. Si votre saisie n'apparaît pas, choisissez Rafraîchir.

        Pour créer une nouvelle entrée, choisissez Create IoT Events input. Renseignez les champs suivants :
        - Nom d'entrée. Saisissez `PressureInput`.
        - Description. Ce nom est facultatif.
        - Téléchargez un fichier JSON. Téléchargez une copie de votre fichier JSON. Cet écran contient un lien vers un exemple de fichier, si vous n'en avez pas. Le code inclut :

```
{
```

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

- Choisissez les attributs d'entrée. Sélectionnez la ou les options appropriées.
- Tags (Balises). Ce nom est facultatif.

Choisissez Créer.

Retournez à l'écran Créer une règle et actualisez le champ Nom de saisie. Sélectionnez l'entrée que vous venez de créer.

b. Mode Batch. Ce nom est facultatif. Si la charge utile est un tableau de messages, sélectionnez cette option.

c. Identifiant du message. Cette action est facultative, mais recommandée.

d. Rôle IAM. Sélectionnez le rôle approprié dans la liste. Si le rôle n'est pas répertorié, choisissez Create new role.

Entrez un nom de rôle et choisissez Create.

Pour ajouter une autre règle, choisissez Ajouter une action de règle

- Action d'erreur. Cette section est facultative. Pour ajouter une action, choisissez Ajouter une action d'erreur et sélectionnez l'action appropriée dans la liste.

Complétez les champs qui s'affichent.

- Choisissez Suivant.

4. Étape 4. Révisez et créez. Passez en revue les informations affichées à l'écran et choisissez Créer.

4. Dans le volet de navigation de gauche, sous Test, choisissez le client de test MQTT.

5. Choisissez Publier dans une rubrique. Renseignez les champs suivants :

- Nom du sujet Entrez un nom pour identifier le message, par exemple`motors/Fulton-A32/status`.
- Charge utile des messages. Saisissez :

```
{
```

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

> **Note**
>
> Modifiez le `messageId` chaque fois que vous publiez un nouveau message.

6. Pour Publier, conservez le même sujet, mais remplacez la `"pressure"` charge utile par une valeur supérieure à la valeur seuil que vous avez spécifiée dans le modèle de détecteur (par exemple**85**).
7. Choisissez Publier.

L'instance de détecteur que vous avez créée génère et vous envoie un message Amazon SNS. Continuez à envoyer des messages avec des relevés de pression supérieurs ou inférieurs au seuil de pression (70 pour cet exemple) pour voir le détecteur en fonctionnement.

Dans cet exemple, vous devez envoyer trois messages avec des relevés de pression inférieurs au seuil pour revenir à l'état normal et recevoir un message Amazon SNS indiquant que la surpression est terminée. Une fois revenu à l'état normal, un message indiquant une pression supérieure à la limite fait passer le détecteur à l'état dangereux et envoie un message Amazon SNS indiquant cette condition.

Maintenant que vous avez créé un modèle d'entrée et de détecteur simple, essayez ce qui suit.

- Consultez d'autres exemples de modèles de détecteurs (modèles) sur la console.
- Suivez les étapes décrites Créez un AWS IoT Events détecteur pour deux états à l'aide de la CLI pour créer un modèle d'entrée et de détecteur à l'aide du AWS CLI
- Découvrez les détails de ce qui Expressions pour filtrer, transformer et traiter les données d'événements est utilisé dans les événements.
- En savoir plus sur Actions prises en charge pour recevoir des données et déclencher des actions dans AWS IoT Events.
- Si quelque chose ne fonctionne pas, voyezRésolution des problèmes AWS IoT Events.

Les meilleures pratiques pour AWS IoT Events

Suivez ces bonnes pratiques pour en tirer le meilleur parti AWS IoT Events.

Rubriques

- Activer la CloudWatch journalisation Amazon lors du développement AWS IoT Events de modèles de détecteurs
- Publiez régulièrement pour enregistrer votre modèle de détecteur lorsque vous travaillez dans la AWS IoT Events console

Activer la CloudWatch journalisation Amazon lors du développement AWS IoT Events de modèles de détecteurs

Amazon CloudWatch surveille vos AWS ressources et les applications que vous utilisez AWS en temps réel. Vous bénéficiez CloudWatch ainsi d'une visibilité à l'échelle du système sur l'utilisation des ressources, les performances des applications et la santé opérationnelle. Lorsque vous développez ou déboguez un modèle de AWS IoT Events détecteur, CloudWatch cela vous aide à savoir ce qui AWS IoT Events se passe et à identifier les erreurs qu'il rencontre.

Pour activer CloudWatch

1. Si ce n'est pas déjà fait, suivez les étapes Configuration des autorisations pour AWS IoT Events ci-dessous pour créer un rôle auquel est attachée une politique autorisant la création et la gestion CloudWatch des journaux pour AWS IoT Events.
2. Accédez à la console AWS IoT Events.
3. Dans le panneau de navigation, sélectionnez Settings (Paramètres).
4. Sur la page Paramètres, choisissez Modifier.
5. Sur la page Modifier les options de journalisation, dans la section Options de journalisation, procédez comme suit :

    a. Pour le niveau de verbosité, sélectionnez une option.

    b. Pour Sélectionner un rôle, sélectionnez un rôle doté des autorisations suffisantes pour effectuer les actions de journalisation que vous avez choisies.

    c. (Facultatif) Si vous avez choisi Debug pour le niveau de verbosité, vous pouvez ajouter des cibles de débogage en procédant comme suit :

i. Sous Cibles de débogage, choisissez Ajouter une option de modèle.

ii. Entrez un nom de modèle de détecteur et (facultatif) KeyValuepour spécifier les modèles de détecteurs et les détecteurs spécifiques (instances) à enregistrer.

6. Choisissez Mettre à jour.

Vos options de journalisation ont été mises à jour avec succès.

Publiez régulièrement pour enregistrer votre modèle de détecteur lorsque vous travaillez dans la AWS IoT Events console

Lorsque vous utilisez la AWS IoT Events console, votre travail en cours est enregistré localement dans votre navigateur. Cependant, vous devez choisir Publier pour enregistrer votre modèle de détecteur dans AWS IoT Events. Après avoir publié un modèle de détecteur, votre travail publié sera disponible dans tous les navigateurs que vous utilisez pour accéder à votre compte.

> **Note**
>
> Si vous ne publiez pas votre œuvre, elle ne sera pas enregistrée. Une fois que vous avez publié un modèle de détecteur, vous ne pouvez pas modifier son nom. Vous pouvez toutefois continuer à modifier sa définition.

Tutoriels pour les cas d' AWS IoT Events utilisation

AWS IoT Events les didacticiels fournissent un ensemble de procédures couvrant divers aspects AWS IoT Events, de la configuration de base à des cas d'utilisation plus spécifiques. Chaque didacticiel présente des exemples de scénarios pratiques, qui vous aident à acquérir des compétences pratiques en matière de création de modèles de détecteurs, de configuration des entrées, de configuration d'actions et d'intégration à d'autres AWS services pour créer de puissantes solutions IoT.

Ce chapitre explique comment :

- Obtenez de l'aide pour choisir les états à inclure dans votre modèle de détecteur et déterminer si vous avez besoin d'une ou de plusieurs instances de détecteur.
- Suivez un exemple qui utilise le AWS CLI.
- Créez une entrée pour recevoir les données de télémétrie d'un appareil et un modèle de détecteur pour surveiller et signaler l'état de l'appareil qui envoie ces données.
- Passez en revue les restrictions et les limitations relatives aux entrées, aux modèles de détecteurs et au AWS IoT Events service.
- Voir un exemple plus complexe de modèle de détecteur, avec commentaires inclus.

Rubriques

- Utilisation AWS IoT Events pour surveiller vos appareils IoT
- Créez un AWS IoT Events détecteur pour deux états à l'aide de la CLI
- AWS IoT Events restrictions et limites du modèle de détecteur
- Exemple commenté : contrôle de la température HVAC avec AWS IoT Events

Utilisation AWS IoT Events pour surveiller vos appareils IoT

Vous pouvez l'utiliser AWS IoT Events pour surveiller vos appareils ou vos processus et prendre des mesures en fonction d'événements importants. Pour ce faire, suivez les étapes de base suivantes :

Création d'entrées

Vous devez disposer d'un moyen pour vos appareils et processus d'y introduire des données de télémétrie. AWS IoT Events Pour ce faire, vous envoyez des messages sous forme d'entrées

à AWS IoT Events. Vous pouvez envoyer des messages sous forme d'entrées de différentes manières :

- Utilisez l' BatchPutMessageopération.
- Définissez une `iotEvents`rule-action pour le moteur de règles.AWS IoT Core La rule-action transfère les données des messages de votre entrée vers. AWS IoT Events
- Dans AWS IoT Analytics, utilisez l'CreateDatasetopération pour créer un ensemble de données avec`contentDeliveryRules`. Ces règles spécifient l' AWS IoT Events entrée à laquelle le contenu de l'ensemble de données est envoyé automatiquement.
- Définissez une action IoT Events dans un modèle `onExit` ou `transitionEvents` un événement de AWS IoT Events `onInput` détecteur. Les informations relatives à l'instance du modèle de détecteur et à l'événement à l'origine de l'action sont renvoyées au système sous forme d'entrée sous le nom que vous spécifiez.

Avant que vos appareils ne commencent à envoyer des données de cette manière, vous devez définir une ou plusieurs entrées. Pour ce faire, attribuez un nom à chaque entrée et spécifiez les champs des données des messages entrants surveillés par l'entrée. AWS IoT Events reçoit ses entrées, sous forme de charge utile JSON, de nombreuses sources. Chaque entrée peut être traitée seule ou combinée à d'autres entrées pour détecter des événements plus complexes.

Création d'un modèle de détecteur

Définissez un modèle de détecteur (un modèle de votre équipement ou de votre processus) à l'aide des états. Pour chaque état, vous définissez une logique conditionnelle (booléenne) qui évalue les entrées entrantes afin de détecter les événements importants. Lorsqu'un événement est détecté, il peut changer d'état ou lancer des actions personnalisées ou prédéfinies à l'aide d'autres AWS services. Vous pouvez définir des événements supplémentaires qui déclenchent des actions lors de l'entrée ou de la sortie d'un état et, éventuellement, lorsqu'une condition est remplie.

Dans ce didacticiel, vous envoyez un message Amazon SNS en tant qu'action lorsque le modèle entre ou sort d'un certain état.

Surveiller un appareil ou un processus

Si vous surveillez plusieurs appareils ou processus, vous spécifiez un champ dans chaque entrée qui identifie le périphérique ou le processus dont provient l'entrée. (Voir le `key` champ dans`CreateDetectorModel`.) Lorsqu'un nouvel appareil est identifié (une nouvelle valeur apparaît dans le champ de saisie identifié par le`key`), un détecteur est créé. (Chaque détecteur est une instance du modèle de détecteur.) Le nouveau détecteur continue ensuite de répondre

aux entrées provenant de cet appareil jusqu'à ce que son modèle de détecteur soit mis à jour ou supprimé.

Si vous surveillez un seul processus (même si plusieurs appareils ou sous-processus envoient des entrées), vous ne spécifiez pas de `key` champ d'identification unique. Dans ce cas, un seul détecteur (instance) est créé lorsque la première entrée arrive.

Envoyez des messages sous forme d'entrées à votre modèle de détecteur

Il existe plusieurs manières d'envoyer un message à partir d'un appareil ou de le traiter en tant qu'entrée dans un AWS IoT Events détecteur sans que vous ayez à appliquer un formatage supplémentaire au message. Dans ce didacticiel, vous allez utiliser la AWS IoT console pour écrire une règle AWS IoT Events d'action pour le moteur de AWS IoT Core règles qui transmet les données de vos messages AWS IoT Events. Pour ce faire, vous devez identifier l'entrée par son nom. Ensuite, vous continuez à utiliser la AWS IoT console pour générer des messages qui sont transférés en tant qu'entrées vers AWS IoT Events.

Comment savoir de quels états vous avez besoin dans un modèle de détecteur ?

Pour déterminer les états que doit présenter votre modèle de détecteur, déterminez d'abord les mesures que vous pouvez prendre. Par exemple, si votre voiture fonctionne à l'essence, vous examinez la jauge de carburant lorsque vous commencez un voyage pour voir si vous avez besoin de faire le plein. Ici, vous n'avez qu'une chose à faire : dites au chauffeur d' « aller chercher de l'essence ». Votre modèle de détecteur a besoin de deux états : « la voiture n'a pas besoin de carburant » et « la voiture a besoin de carburant ». En général, vous souhaitez définir un état pour chaque action possible, plus un autre lorsqu'aucune action n'est requise. Cela fonctionne même si l'action elle-même est plus compliquée. Par exemple, vous pouvez rechercher et inclure des informations indiquant où trouver la station-service la plus proche ou le prix le moins cher, mais vous le faites lorsque vous envoyez le message « allez chercher de l'essence ».

Pour décider dans quel état entrer ensuite, vous examinez les entrées. Les entrées contiennent les informations dont vous avez besoin pour décider dans quel état vous devez vous trouver. Pour créer une saisie, vous devez sélectionner un ou plusieurs champs dans un message envoyé par votre appareil ou processus pour vous aider à prendre une décision. Dans cet exemple, vous avez besoin d'une entrée qui indique le niveau de carburant actuel (« pourcentage de remplissage »). Peut-être que votre voiture vous envoie plusieurs messages différents, chacun contenant plusieurs champs différents. Pour créer cette entrée, vous devez sélectionner le message et le champ indiquant le

niveau actuel de la jauge de gaz. La durée du voyage que vous vous apprêtez à effectuer (« distance jusqu'à la destination ») peut être codée en dur pour simplifier les choses ; vous pouvez utiliser la durée moyenne de votre trajet. Vous allez effectuer des calculs en fonction de l'entrée (à combien de gallons correspond ce pourcentage plein ? est la durée moyenne du trajet supérieure aux miles que vous pouvez parcourir, compte tenu des gallons dont vous disposez et de votre moyenne (« miles par gallon »). Vous effectuez ces calculs et vous envoyez des messages lors d'événements.

Jusqu'à présent, vous avez deux états et une entrée. Vous avez besoin d'un événement dans le premier état qui effectue les calculs en fonction de l'entrée et décide de passer au second état. Il s'agit d'un événement de transition. (`transitionEvents`figurent dans la liste des `onInput` événements d'un État. À la réception d'une entrée dans ce premier état, l'événement effectue une transition vers le second état, si celui de l'événement `condition` est satisfait.) Lorsque vous atteignez le deuxième état, vous envoyez le message dès que vous entrez dans l'état. (Vous utilisez un `onEnter` événement. Lorsque vous entrez dans le deuxième état, cet événement envoie le message. Pas besoin d'attendre l'arrivée d'une autre entrée.) Il existe d'autres types d'événements, mais c'est tout ce dont vous avez besoin pour un exemple simple.

Les autres types d'événements sont `onExit` et`onInput`. Dès qu'une entrée est reçue et que la condition est remplie, un `onInput` événement exécute les actions spécifiées. Lorsqu'une opération quitte son état actuel et que la condition est remplie, l'`onExit`événement exécute les actions spécifiées.

Il vous manque quelque chose ? Oui, comment revenir au premier état « la voiture n'a pas besoin de carburant » ? Une fois que vous avez rempli votre réservoir d'essence, l'entrée indique qu'il est plein. Dans votre deuxième état, vous avez besoin d'un événement de transition vers le premier état qui se produit lorsque l'entrée est reçue (dans les `onInput:` événements du second état). Il devrait revenir au premier état si ses calculs indiquent que vous avez maintenant assez d'essence pour vous rendre là où vous voulez aller.

C'est l'essentiel. Certains modèles de détecteurs deviennent plus complexes en ajoutant des états qui reflètent des entrées importantes, et pas seulement des actions possibles. Par exemple, un modèle de détecteur peut avoir trois états permettant de suivre la température : un état « normal », un état « trop chaud » et un état « problème potentiel ». Vous passez à l'état potentiellement problématique lorsque la température dépasse un certain niveau, mais qu'elle n'est pas encore trop chaude. Vous ne voulez pas envoyer d'alarme à moins qu'elle ne reste à cette température pendant plus de 15 minutes. Si la température revient à la normale avant cette date, le détecteur revient à l'état normal. Si le chronomètre expire, le détecteur passe à l'état trop chaud et envoie une alarme, par mesure de prudence. Vous pouvez faire la même chose en utilisant des variables et un ensemble

plus complexe de conditions d'événement. Mais il est souvent plus facile d'utiliser un autre état pour, en fait, stocker les résultats de vos calculs.

Comment savoir si vous avez besoin d'une ou de plusieurs instances de détecteur ?

Pour déterminer le nombre d'instances dont vous avez besoin, demandez-vous « Qu'est-ce qui vous intéresse de savoir ? » Imaginons que vous vouliez connaître le temps qu'il fait aujourd'hui. Est-ce qu'il pleut (État) ? Avez-vous besoin d'un parapluie (action) ? Vous pouvez avoir un capteur qui indique la température, un autre qui indique l'humidité et d'autres qui signalent la pression barométrique, la vitesse et la direction du vent, ainsi que les précipitations. Mais vous devez surveiller tous ces capteurs ensemble pour déterminer l'état de la météo (pluie, neige, ciel couvert, soleil) et les mesures appropriées à prendre (prendre un parapluie ou appliquer un écran solaire). Malgré le nombre de capteurs, vous souhaitez qu'une instance de détecteur surveille l'état de la météo et vous indique les mesures à prendre.

Mais si vous êtes le prévisionniste météo de votre région, il se peut que vous disposiez de plusieurs instances de tels réseaux de capteurs, situées à différents endroits de la région. Les gens de chaque endroit doivent connaître le temps qu'il fait à cet endroit. Dans ce cas, vous avez besoin de plusieurs instances de votre détecteur. Les données rapportées par chaque capteur à chaque emplacement doivent inclure un champ que vous avez désigné comme `key` champ. Ce champ permet AWS IoT Events de créer une instance de détecteur pour la zone, puis de continuer à acheminer ces informations vers cette instance de détecteur au fur et à mesure qu'elles arrivent. Fini les cheveux abîmés ou les coups de soleil sur le nez !

Essentiellement, vous avez besoin d'une instance de détecteur si vous avez une situation (un processus ou un emplacement) à surveiller. Si vous devez surveiller de nombreuses situations (sites, processus), vous avez besoin de plusieurs instances de détecteurs.

Créez un AWS IoT Events détecteur pour deux états à l'aide de la CLI

Dans cet exemple, nous appelons les AWS CLI commandes AWS IoT Events APIs using pour créer un détecteur qui modélise deux états d'un moteur : un état normal et un état de surpression.

Lorsque la pression mesurée dans le moteur dépasse un certain seuil, le modèle passe en état de surpression et envoie un message Amazon Simple Notification Service (Amazon SNS) pour avertir le

technicien de cette situation. Lorsque la pression chute en dessous du seuil pendant trois relevés de pression consécutifs, le modèle revient à l'état normal et envoie un autre message Amazon SNS pour confirmer que le problème est résolu. Nous avons besoin de trois mesures consécutives en dessous du seuil de pression afin d'éliminer le bégaiement éventuel dû à des messages de surpression ou à des messages normaux en cas de phase de reprise non linéaire ou de lecture anormale ponctuelle.

Vous trouverez ci-dessous un aperçu des étapes de création du détecteur.

Créez des entrées.

Pour surveiller vos appareils et processus, vous devez pouvoir obtenir des données télémétriques dans AWS IoT Events. Cela se fait en envoyant des messages en tant qu'entrées à AWS IoT Events. Pour ce faire, plusieurs options s'offrent à vous :

- Utilisez l' BatchPutMessageopération. Cette méthode est simple mais nécessite que vos appareils ou processus puissent accéder à l' AWS IoT Events API via un SDK ou le AWS CLI.
- Dans AWS IoT Core, écrivez une règle AWS IoT Events d'action pour le moteur de AWS IoT Core règles qui transfère les données de vos messages vers AWS IoT Events. Cela identifie l'entrée par son nom. Utilisez cette méthode si vos appareils ou processus peuvent envoyer des messages ou le font déjà AWS IoT Core. Cette méthode nécessite généralement moins de puissance de calcul de la part d'un appareil.
- Dans AWS IoT Analytics, utilisez l' CreateDatasetopération pour créer un ensemble de données spécifiant `contentDeliveryRules` l' AWS IoT Events entrée, où le contenu de l'ensemble de données est envoyé automatiquement. Utilisez cette méthode si vous souhaitez contrôler vos appareils ou vos processus en fonction des données agrégées ou analysées dans AWS IoT Analytics.

Avant que vos appareils puissent envoyer des données de cette manière, vous devez définir une ou plusieurs entrées. Pour ce faire, attribuez un nom à chaque entrée et spécifiez les champs des données des messages entrants surveillés par l'entrée.

Création d'un modèle de détecteur

Créez un modèle de détecteur (un modèle de votre équipement ou de votre processus) à l'aide des états. Pour chaque état, définissez une logique conditionnelle (booléenne) qui évalue les entrées entrantes afin de détecter les événements significatifs. Lorsqu'un événement est détecté, il peut changer d'état ou lancer des actions personnalisées ou prédéfinies à l'aide d'autres AWS services. Vous pouvez définir des événements supplémentaires qui déclenchent des actions lors de l'entrée ou de la sortie d'un état et, éventuellement, lorsqu'une condition est remplie.

Surveillez plusieurs appareils ou processus

Si vous surveillez plusieurs appareils ou processus et que vous souhaitez suivre chacun d'eux séparément, spécifiez un champ dans chaque entrée qui identifie le périphérique ou le processus dont provient l'entrée. Voir le `key` champ dans`CreateDetectorModel`. Lorsqu'un nouveau dispositif est identifié (une nouvelle valeur apparaît dans le champ de saisie identifié par le`key`), une instance de détecteur est créée. La nouvelle instance de détecteur continue de répondre aux entrées provenant de cet appareil particulier jusqu'à ce que son modèle de détecteur soit mis à jour ou supprimé. Vous avez autant de détecteurs uniques (instances) qu'il y a de valeurs uniques dans les `key` champs de saisie.

Surveillez un seul appareil ou processus

Si vous surveillez un seul processus (même si plusieurs appareils ou sous-processus envoient des entrées), vous ne spécifiez pas de `key` champ d'identification unique. Dans ce cas, un seul détecteur (instance) est créé lorsque la première entrée arrive. Par exemple, vous pouvez avoir des capteurs de température dans chaque pièce d'une maison, mais une seule unité CVC pour chauffer ou climatiser toute la maison. Vous ne pouvez donc contrôler cela que dans le cadre d'un processus unique, même si chaque occupant de la chambre souhaite que son vote (contribution) l'emporte.

Envoyez des messages à partir de vos appareils ou processus en tant qu'entrées pour votre modèle de détecteur

Nous avons décrit les différentes manières d'envoyer un message à partir d'un appareil ou d'un processus en tant qu'entrée dans un AWS IoT Events détecteur dans les entrées. Après avoir créé les entrées et créé le modèle de détecteur, vous êtes prêt à commencer à envoyer des données.

> **Note**
>
> Lorsque vous créez un modèle de détecteur ou que vous mettez à jour un modèle existant, plusieurs minutes s'écoulent avant que le nouveau modèle de détecteur ou le modèle mis à jour ne commence à recevoir des messages et à créer des détecteurs (instances). Si le modèle de détecteur est mis à jour, il est possible que vous continuiez à observer un comportement basé sur la version précédente pendant cette période.

Rubriques

- Création d'une AWS IoT Events entrée pour capturer les données de l'appareil

- Créez un modèle de détecteur pour représenter les états des appareils dans AWS IoT Events
- Envoyer des messages sous forme d'entrées à un détecteur dans AWS IoT Events

Création d'une AWS IoT Events entrée pour capturer les données de l'appareil

Lorsque vous configurez les entrées pour AWS IoT Events, vous pouvez les utiliser AWS CLI pour définir la manière dont vos appareils communiquent les données des capteurs. Par exemple, si vos appareils envoient des messages au format JSON avec des identifiants de moteur et des relevés de capteurs, vous pouvez capturer ces données en créant une entrée qui mappe des attributs spécifiques à partir des messages, tels que la pression et l'identifiant du moteur. Le processus commence par définir une entrée dans un fichier JSON, en spécifiant les points de données pertinents et en utilisant le AWS CLI pour enregistrer l'entrée pour AWS IoT Events. Cela permet AWS IoT de surveiller et de répondre aux conditions critiques sur la base des données des capteurs en temps réel.

Supposons, par exemple, que vos appareils envoient des messages au format suivant.

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

Vous pouvez créer une entrée pour capturer les `pressure` données et `motorid` (qui identifie le périphérique spécifique qui a envoyé le message) à l'aide de la AWS CLI commande suivante.

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

Le fichier `pressureInput.json` contient les éléments suivants.

```
{
  "inputName": "PressureInput",
  "inputDescription": "Pressure readings from a motor",
  "inputDefinition": {
    "attributes": [
```

```
      { "jsonPath": "sensorData.pressure" },
      { "jsonPath": "motorid" }
    ]
  }
}
```

Lorsque vous créez vos propres entrées, n'oubliez pas de collecter d'abord des exemples de messages sous forme de fichiers JSON à partir de vos appareils ou processus. Vous pouvez les utiliser pour créer une entrée à partir de la console ou de la CLI.

Créez un modèle de détecteur pour représenter les états des appareils dans AWS IoT Events

DansCréation d'une AWS IoT Events entrée pour capturer les données de l'appareil, vous avez créé un message `input` basé sur un message qui rapporte les données de pression d'un moteur. Pour continuer avec l'exemple, voici un modèle de détecteur qui répond à un événement de surpression dans un moteur.

Vous créez deux états : `Normal` « » et « `Dangerous` ». Chaque détecteur (instance) passe à l'état `Normal` « » lors de sa création. L'instance est créée lorsqu'une entrée avec une valeur unique pour le `key` « `motorid` » arrive.

Si l'instance du détecteur reçoit une valeur de pression supérieure ou égale à 70, elle passe à l'état `Dangerous` « » et envoie un message Amazon SNS en guise d'avertissement. Si les relevés de pression reviennent à la normale (moins de 70) pour trois entrées consécutives, le détecteur revient à l'état « `Normal` » et envoie un autre message Amazon SNS en guise de feu vert.

Cet exemple de modèle de détecteur suppose que vous avez créé deux rubriques Amazon SNS dont les noms de ressources Amazon (ARNs) apparaissent dans la définition sous `"targetArn": "arn:aws:sns:us-east-1:123456789012:underPressureAction"` la forme et. `"targetArn": "arn:aws:sns:us-east-1:123456789012:pressureClearedAction"`

Pour plus d'informations, consultez le guide du développeur Amazon Simple Notification Service et, plus précisément, la documentation de l'CreateTopicopération dans le manuel Amazon Simple Notification Service API Reference.

Cet exemple suppose également que vous avez créé un rôle Gestion des identités et des accès AWS (IAM) doté des autorisations appropriées. L'ARN de ce rôle est indiqué dans la définition du modèle de détecteur sous la forme`"roleArn": "arn:aws:iam::123456789012:role/IoTEventsRole"`. Suivez les étapes décrites Configuration des autorisations pour AWS IoT Events

pour créer ce rôle et copiez l'ARN du rôle à l'endroit approprié dans la définition du modèle de détecteur.

Vous pouvez créer le modèle de détecteur à l'aide de la AWS CLI commande suivante.

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

Le fichier "`motorDetectorModel.json`" contient les éléments suivants.

```
{
  "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"
}
```

Envoyer des messages sous forme d'entrées à un détecteur dans AWS IoT Events

Vous avez maintenant défini une entrée qui identifie les champs importants des messages envoyés depuis un appareil (voirCréation d'une AWS IoT Events entrée pour capturer les données de l'appareil). Dans la section précédente, vous avez créé un `detector model` qui répond à un événement de surpression dans un moteur (voirCréez un modèle de détecteur pour représenter les états des appareils dans AWS IoT Events).

Pour compléter l'exemple, envoyez des messages depuis un périphérique (dans ce cas, un ordinateur sur lequel le AWS CLI système est installé) comme entrées au détecteur.

> **Note**
>
> Lorsque vous créez un modèle de détecteur ou que vous mettez à jour un modèle existant, plusieurs minutes s'écoulent avant que le nouveau modèle de détecteur ou le modèle mis à jour ne commence à recevoir des messages et à créer des détecteurs (instances). Si vous mettez à jour le modèle du détecteur, il est possible que vous continuiez à observer un comportement basé sur la version précédente pendant cette période.

Utilisez la AWS CLI commande suivante pour envoyer un message contenant des données dépassant le seuil.

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

Le fichier « `highPressureMessage.json` » contient les éléments suivants.

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

```
}
```

Vous devez modifier le `messageId` dans chaque message envoyé. Si vous ne le modifiez pas, le AWS IoT Events système déduplique les messages. AWS IoT Events ignore un message s'il contient le même message `messageID` qu'un autre envoyé au cours des cinq dernières minutes.

À ce stade, un détecteur (instance) est créé pour surveiller les événements du moteur`"Fulton-A32"`. Ce détecteur entre dans l'`"Normal"`état lorsqu'il est créé. Mais comme nous avons envoyé une valeur de pression supérieure au seuil, elle passe immédiatement à l'`"Dangerous"`état. Ce faisant, le détecteur envoie un message au point de terminaison Amazon SNS dont l'ARN est l'ARN. `arn:aws:sns:us-east-1:123456789012:underPressureAction`

Exécutez la AWS CLI commande suivante pour envoyer un message contenant des données inférieures au seuil de pression.

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

Le fichier `normalPressureMessage.json` contient les éléments suivants.

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

Vous devez le modifier `messageId` dans le fichier chaque fois que vous appelez la `BatchPutMessage` commande dans un délai de cinq minutes. Envoyez le message deux fois de plus. Une fois le message envoyé trois fois, le détecteur (instance) du moteur « `Fulton-A32` » envoie un message au point de terminaison Amazon SNS `"arn:aws:sns:us-east-1:123456789012:pressureClearedAction"` et entre à nouveau dans l'état. `"Normal"`

> **Note**
>
> Vous pouvez envoyer plusieurs messages à la fois avec`BatchPutMessage`. Cependant, l'ordre dans lequel ces messages sont traités n'est pas garanti. Pour garantir que les messages (entrées) sont traités dans l'ordre, envoyez-les un par un et attendez une réponse positive chaque fois que l'API est appelée.

Vous trouverez ci-dessous des exemples de charges utiles de messages SNS créées par l'exemple de modèle de détecteur décrit dans cette section.

sur l'événement « Seuil de pression dépassé »

```
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"
}
```

sur l'événement « Normal Pressure Restaurée »

```
IoT> {
```

```
  "eventTime":1558129925568,
  "payload":{
    "actionExecutionId":"7e25fd38-2533-303d-899f-c979792a12cb",
    "detector":{
      "detectorModelName":"motorDetectorModel",
      "keyValue":"Fulton-A32",
      "detectorModelVersion":"1"
    },
    "eventTriggerDetails":{
      "inputName":"PressureInput",
      "messageId":"00004",
      "triggerType":"Message"
    },
    "state":{
      "stateName":"Dangerous",
      "variables":{
        "pressureThresholdBreached":0
      },
      "timers":{}
    }
  },
  "eventName":"Normal Pressure Restored"
}
```

Si vous avez défini des temporisateurs, leur état actuel est également affiché dans les charges utiles des messages SNS.

Les charges utiles des messages contiennent des informations sur l'état du détecteur (instance) au moment où le message a été envoyé (c'est-à-dire au moment où l'action SNS a été exécutée). Vous pouvez utiliser cette https://docs.aws.amazon.com/iotevents/latest/apireference/API_iotevents-data_DescribeDetector.html opération pour obtenir des informations similaires sur l'état du détecteur.

AWS IoT Events restrictions et limites du modèle de détecteur

Les éléments suivants sont importants à prendre en compte lors de la création d'un modèle de détecteur.

Comment utiliser le **`actions`** terrain

Le `actions` champ est une liste d'objets. Vous pouvez avoir plusieurs objets, mais une seule action est autorisée pour chaque objet.

Example

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

Comment utiliser le **`condition`** terrain

Cela `condition` est obligatoire `transitionEvents` et facultatif dans les autres cas.

Si le `condition` champ n'est pas présent, c'est équivalent à`"condition": true`.

Le résultat de l'évaluation d'une expression de condition doit être une valeur booléenne. Si le résultat n'est pas une valeur booléenne, il est équivalent à la valeur `nextState` spécifiée dans l'`actions`événement `false` et n'initiera pas la transition vers celle-ci.

Disponibilité de valeurs variables

Par défaut, si la valeur d'une variable est définie dans un événement, sa nouvelle valeur n'est pas disponible ou n'est pas utilisée pour évaluer les conditions d'autres événements du même groupe. La nouvelle valeur n'est pas disponible ou utilisée dans une condition d'événement dans le même `onInput` `onExit` champ `onEnter` ou dans le même champ.

Définissez le `evaluationMethod` paramètre dans la définition du modèle de détecteur pour modifier ce comportement. Lorsque le paramètre `evaluationMethod` est défini sur`SERIAL`, les variables sont mises à jour et les conditions des événements sont évaluées dans l'ordre dans lequel les événements sont définis. Sinon, lorsque cette valeur `evaluationMethod` est définie `BATCH` ou est définie par défaut, les variables d'un état sont mises à jour et les événements d'un état ne sont exécutés qu'une fois que toutes les conditions de l'événement ont été évaluées.

Dans l'"Dangerous"état, onInput sur le terrain, "$variable.pressureThresholdBreached" est décrémenté de un dans le "Pressure Okay" cas où la condition est remplie (lorsque la pression d'entrée du courant est inférieure ou égale à 70).

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

Le détecteur doit revenir à l'"Normal"état lorsqu'il "$variable.pressureThresholdBreached" atteint 0 (c'est-à-dire lorsqu'il a reçu trois relevés de pression contigus inférieurs ou égaux à 70). L'"BackToNormal"événement in transitionEvents doit tester une valeur inférieure ou égale à 1 (et non à 0), et également vérifier à nouveau que la valeur actuelle donnée par "$input.PressureInput.sensorData.pressure" est inférieure ou égale à 70. "$variable.pressureThresholdBreached"

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

Sinon, si la condition teste uniquement la valeur de la variable, deux lectures normales suivies d'une lecture de surpression rempliront la condition et reviendront à l'"Normal"état. La condition examine la valeur qui "$variable.pressureThresholdBreached" a été

donnée lors du traitement précédent d'une entrée. La valeur de la variable est remise à 3 dans l'"`0verpressurized`"événement, mais n'oubliez pas que cette nouvelle valeur n'est encore disponible pour personne`condition`.

Par défaut, chaque fois qu'un contrôle entre dans le `onInput` champ, a `condition` peut uniquement voir la valeur d'une variable telle qu'elle était au début du traitement de l'entrée, avant qu'elle ne soit modifiée par les actions spécifiées dans`onInput`. Il en va de même pour `onEnter` et`onExit`. Toute modification apportée à une variable lorsque nous entrons ou quittons l'état n'est pas disponible pour les autres conditions spécifiées dans le même `onEnter` ou `onExit` les champs.

Latence lors de la mise à jour d'un modèle de détecteur

Si vous mettez à jour, supprimez et recréez un modèle de détecteur (voir UpdateDetectorModel), il y a un certain délai avant que tous les détecteurs générés (instances) soient supprimés et que le nouveau modèle soit utilisé pour recréer les détecteurs. Ils sont recréés une fois que le nouveau modèle de détecteur prend effet et que de nouvelles entrées arrivent. Pendant ce temps, les entrées peuvent continuer à être traitées par les détecteurs générés par la version précédente du modèle de détecteur. Pendant cette période, il est possible que vous continuiez à recevoir des alertes définies par le modèle de détecteur précédent.

Espaces dans les touches de saisie

Les espaces sont autorisés dans les touches de saisie, mais les références à la clé doivent être encadrées par des backticks, à la fois dans la définition de l'attribut d'entrée et lorsque la valeur de la clé est référencée dans une expression. Par exemple, si la charge utile d'un message est la suivante :

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

Utilisez ce qui suit pour définir l'entrée.

```
{
  "inputName": "PressureInput",
  "inputDescription": "Pressure readings from a motor",
```

```
  "inputDefinition": {
    "attributes": [
      { "jsonPath": "sensorData.`motor pressure`" },
      { "jsonPath": "`motor id`" }
    ]
  }
}
```

Dans une expression conditionnelle, vous devez également faire référence à la valeur d'une telle clé en utilisant des backticks.

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

Exemple commenté : contrôle de la température HVAC avec AWS IoT Events

Certains des exemples de fichiers JSON suivants contiennent des commentaires en ligne, ce qui les rend non valides. Les versions complètes de ces exemples, sans commentaires, sont disponibles à l'adresseExemple : utilisation du contrôle de température HVAC avec AWS IoT Events.

Cet exemple implémente un modèle de commande de thermostat qui vous permet d'effectuer les opérations suivantes.

- Définissez un seul modèle de détecteur pouvant être utilisé pour surveiller et contrôler plusieurs zones. Une instance de détecteur est créée pour chaque zone.
- Ingérez les données de température provenant de plusieurs capteurs dans chaque zone de contrôle.
- Modifiez le point de réglage de température d'une zone.
- Définissez les paramètres opérationnels pour chaque zone et réinitialisez ces paramètres pendant que l'instance est en cours d'utilisation.
- Ajoutez ou supprimez des capteurs de manière dynamique dans une zone.
- Spécifiez une durée de fonctionnement minimale pour protéger les unités de chauffage et de refroidissement.
- Refusez les lectures anormales du capteur.

- Définissez des points de consigne d'urgence qui déclenchent immédiatement le chauffage ou le refroidissement si l'un des capteurs signale une température supérieure ou inférieure à un seuil donné.
- Signalez les relevés anormaux et les pics de température.

Rubriques

- Définitions d'entrée pour les modèles de détecteurs dans AWS IoT Events
- Création d'une définition AWS IoT Events de modèle de détecteur
- BatchUpdateDetector À utiliser pour mettre à jour un modèle AWS IoT Events de détecteur
- À utiliser BatchPutMessage pour les entrées dans AWS IoT Events
- Ingérez des messages MQTT dans AWS IoT Events
- Générez des messages Amazon SNS dans AWS IoT Events
- Configurez l' DescribeDetector API dans AWS IoT Events
- Utilisez le moteur de AWS IoT Core règles pour AWS IoT Events

Définitions d'entrée pour les modèles de détecteurs dans AWS IoT Events

Nous voulons créer un modèle de détecteur que nous pouvons utiliser pour surveiller et contrôler la température dans plusieurs zones différentes. Chaque zone peut être équipée de plusieurs capteurs qui signalent la température. Nous supposons que chaque zone est desservie par une unité de chauffage et une unité de refroidissement qui peuvent être activées ou désactivées pour contrôler la température de la zone. Chaque zone est contrôlée par une instance de détecteur.

Étant donné que les différentes zones que nous surveillons et contrôlons peuvent avoir des caractéristiques différentes nécessitant des paramètres de contrôle différents, nous les définissons `'seedTemperatureInput'` de manière à fournir ces paramètres pour chaque zone. Lorsque nous envoyons l'un de ces messages d'entrée à AWS IoT Events, une nouvelle instance de modèle de détecteur contenant les paramètres que nous voulons utiliser dans cette zone est créée. Voici la définition de cette entrée.

Commande de la CLI :

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

Dossier : `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" }
    ]
  }
}
```

Réponse :

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

Remarques

- Une nouvelle instance de détecteur est créée pour chaque message unique `'areaId'` reçu. Voir le `'key'` champ dans la `'areaDetectorModel'` définition.
- La température moyenne peut varier `'desiredTemperature'` par rapport à `'allowedError'` avant que les unités de chauffage ou de refroidissement ne soient activées pour la zone.

- Si un capteur signale une température supérieure à`'rangeHigh'`, le détecteur signale un pic et démarre immédiatement l'unité de refroidissement.
- Si un capteur signale une température inférieure à`'rangeLow'`, le détecteur signale un pic et démarre immédiatement l'unité de chauffage.
- Si un capteur signale une température supérieure `'anomalousHigh'` ou inférieure à`'anomalousLow'`, le détecteur signale une lecture anormale du capteur, mais ignore la lecture de température signalée.
- `'sensorCount'`Indique au détecteur le nombre de capteurs signalés pour la zone. Le détecteur calcule la température moyenne dans la zone en attribuant le facteur de pondération approprié à chaque mesure de température qu'il reçoit. De ce fait, le détecteur n'aura pas à suivre ce que chaque capteur signale, et le nombre de capteurs peut être modifié dynamiquement, selon les besoins. Cependant, si un capteur individuel est déconnecté, le détecteur ne le saura pas ou n'en tiendra pas compte. Nous vous recommandons de créer un autre modèle de détecteur spécialement pour surveiller l'état de connexion de chaque capteur. Le fait de disposer de deux modèles de détecteurs complémentaires simplifie la conception des deux.
- La `'noDelay'` valeur peut être `true` ou`false`. Une fois qu'une unité de chauffage ou de refroidissement est allumée, elle doit rester allumée pendant un certain temps minimum afin de protéger l'intégrité de l'unité et de prolonger sa durée de vie. Si cette valeur `'noDelay'` est définie sur`false`, l'instance du détecteur applique un délai avant d'éteindre les unités de refroidissement et de chauffage, afin de garantir qu'elles fonctionnent le moins longtemps possible. Le nombre de secondes de retard a été codé en dur dans la définition du modèle de détecteur car nous ne sommes pas en mesure d'utiliser une valeur variable pour régler un temporisateur.

Le `'temperatureInput'` est utilisé pour transmettre les données du capteur à une instance de détecteur.

Commande de la CLI :

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

Dossier : `temperatureInput.json`

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

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

Réponse :

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

Remarques

- Le `'sensorId'` n'est pas utilisé par un exemple d'instance de détecteur pour contrôler ou surveiller directement un capteur. Il est automatiquement transmis aux notifications envoyées par l'instance du détecteur. À partir de là, il peut être utilisé pour identifier les capteurs défaillants (par exemple, un capteur qui envoie régulièrement des mesures anormales peut être sur le point de tomber en panne) ou qui sont hors ligne (lorsqu'il est utilisé comme entrée pour un modèle de détecteur supplémentaire qui surveille le rythme cardiaque de l'appareil). Ils `'sensorId'` peuvent également aider à identifier les zones chaudes ou froides d'une zone si ses valeurs diffèrent régulièrement de la moyenne.
- Le `'areaId'` est utilisé pour acheminer les données du capteur vers l'instance de détecteur appropriée. Une instance de détecteur est créée pour chaque message unique `'areaId'` reçu. Voir le `'key'` champ dans la `'areaDetectorModel'` définition.

Création d'une définition AWS IoT Events de modèle de détecteur

L'`'areaDetectorModel'`exemple contient des commentaires en ligne.

Commande de la CLI :

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

Dossier : `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"
}
```

Réponse :

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

```
        "detectorModelVersion": "1"
    }
}
```

BatchUpdateDetector À utiliser pour mettre à jour un modèle AWS IoT Events de détecteur

Vous pouvez utiliser cette `BatchUpdateDetector` opération pour placer une instance de détecteur dans un état connu, y compris le temporisateur et les valeurs des variables. Dans l'exemple suivant, l'`BatchUpdateDetector`opération réinitialise les paramètres opérationnels d'une zone soumise à une surveillance et à un contrôle de température. Cette opération vous permet de le faire sans avoir à supprimer, recréer ou mettre à jour le modèle du détecteur.

Commande de la CLI :

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

Dossier : `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": [
        ]
      }
    }
  ]
}
```

Réponse :

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

À utiliser BatchPutMessage pour les entrées dans AWS IoT Events

Example 1

Utilisez cette `BatchPutMessage` opération pour envoyer un `"seedTemperatureInput"` message qui définit les paramètres opérationnels pour une zone donnée sous contrôle et surveillance de la température. Tout message reçu par AWS IoT Events celui-ci contient un nouveau `"areaId"` provoque la création d'une nouvelle instance de détecteur. Mais la nouvelle instance de détecteur ne changera pas d'état `"idle"` et ne commencera pas à surveiller la température et à contrôler les unités de chauffage ou de refroidissement tant qu'un `"seedTemperatureInput"` message n'aura pas été reçu pour la nouvelle zone.

Commande de la CLI :

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

Dossier : `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}"
    }
  ]
}
```

Réponse :

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

Example

2

Utilisez cette `BatchPutMessage` opération pour envoyer un `"temperatureInput"` message afin de signaler les données du capteur de température d'un capteur dans une zone de contrôle et de surveillance donnée.

Commande de la CLI :

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

Dossier : `temperatureExample.json`

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

Réponse :

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

Example 3

Utilisez cette `BatchPutMessage` opération pour envoyer un `"seedTemperatureInput"` message afin de modifier la valeur de la température souhaitée pour une zone donnée.

Commande de la CLI :

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

Dossier : `seedSetDesiredTemp.json`

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

Réponse :

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

Ingérez des messages MQTT dans AWS IoT Events

Si les ressources informatiques de vos capteurs ne peuvent pas utiliser l'`"BatchPutMessage"`API, mais peuvent envoyer leurs données au courtier de AWS IoT Core messages à l'aide d'un client MQTT léger, vous pouvez créer une règle de AWS IoT Core sujet pour rediriger les données des messages vers une AWS IoT Events entrée. Ce qui suit est une définition d'une règle de AWS

IoT Events sujet qui prend les champs de "sensorId" saisie "areaId" et du sujet MQTT, et le "sensorData.temperature" champ du champ de charge utile du message, et ingère ces données "temp" dans notre. AWS IoT Events "temperatureInput"

Commande de la CLI :

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

Dossier : 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"
  }
}
```

Réponse : [aucune]

Si le capteur envoie un message sur le sujet "update/temperature/Area51/03" avec la charge utile suivante.

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

Cela entraîne l'ingestion de données AWS IoT Events comme si l'appel d'"BatchPutMessage"API suivant avait été effectué.

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

Dossier : `spoofExample.json`

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

Générez des messages Amazon SNS dans AWS IoT Events

Voici des exemples de messages SNS générés par l'instance du `"Area51"` détecteur.

AWS IoT Events peut s'intégrer à Amazon SNS pour générer et publier des notifications en fonction des événements détectés. Cette section explique comment une instance de AWS IoT Events détecteur, en particulier le détecteur « Area51 », génère des messages Amazon SNS. Ces exemples présentent la structure et le contenu des notifications Amazon SNS déclenchées par différents états et événements au sein du AWS IoT Events détecteur, illustrant ainsi la puissance de la combinaison avec Amazon AWS IoT Events SNS pour les alertes et les communications en temps réel.

```
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"}
```

Configurez l' DescribeDetector API dans AWS IoT Events

L'`DescribeDetector`API in vous AWS IoT Events permet de récupérer des informations détaillées sur une instance de détecteur spécifique. Cette opération fournit des informations sur l'état actuel, les valeurs variables et les temporisateurs actifs d'un détecteur. En utilisant cette API, vous pouvez surveiller l'état en temps réel de vos AWS IoT Events détecteurs, ce qui facilite le débogage, l'analyse et la gestion de vos flux de travail de traitement des événements IoT.

Commande de la CLI :

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

Réponse :

```
{
    "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"
    }
}
```

Utilisez le moteur de AWS IoT Core règles pour AWS IoT Events

Les règles suivantes republient les messages AWS IoT Core MQTT sous forme de messages de demande de mise à jour instantanée. Nous supposons que AWS IoT Core les éléments sont définis pour une unité de chauffage et une unité de refroidissement pour chaque zone contrôlée par le modèle de détecteur. Dans cet exemple, nous avons défini des éléments nommés `"Area51HeatingUnit"` et`"Area51CoolingUnit"`.

Commande de la CLI :

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

Dossier : `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"
        }
      }
    ]
  }
}
```

Réponse : [vide]

Commande de la CLI :

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

Dossier : `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"
        }
      }
    ]
  }
}
```

Réponse : [vide]

Commande de la CLI :

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

Dossier : 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"
        }
      }
    ]
  }
}
```

Réponse : [vide]

Commande de la CLI :

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

Dossier : 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"
        }
      }
    ]
  }
}
```

Réponse : [vide]

Actions prises en charge pour recevoir des données et déclencher des actions dans AWS IoT Events

AWS IoT Events peut déclencher des actions lorsqu'il détecte un événement spécifique ou un événement de transition. Vous pouvez définir des actions intégrées pour utiliser un temporisateur, définir une variable ou envoyer des données à d'autres AWS ressources. Découvrez comment configurer et personnaliser ces actions afin de créer des réponses automatisées à vos différents événements liés à l'IoT.

> **Note**
>
> Lorsque vous définissez une action dans un modèle de détecteur, vous pouvez utiliser des expressions pour les paramètres de type chaîne de données. Pour plus d'informations, consultez Expressions.

AWS IoT Events prend en charge les actions suivantes qui vous permettent d'utiliser un temporisateur ou de définir une variable :

- `setTimer`pour créer un chronomètre.
- `resetTimer`pour réinitialiser le chronomètre.
- `clearTimer`pour supprimer le chronomètre.
- `setVariable`pour créer une variable.

AWS IoT Events prend en charge les actions suivantes qui vous permettent de travailler avec AWS les services :

- `iotTopicPublish`pour publier un message sur un sujet MQTT.
- `iotEvents`pour envoyer des données en AWS IoT Events tant que valeur d'entrée.
- `iotSiteWise` pour envoyer des données à une propriété de ressources dans AWS IoT SiteWise.
- `dynamoDB`pour envoyer des données vers une table Amazon DynamoDB.
- `dynamoDBv2`pour envoyer des données vers une table Amazon DynamoDB.
- `firehose`pour envoyer des données vers un flux Amazon Data Firehose.

- `lambda`pour invoquer une AWS Lambda fonction.
- `sns`pour envoyer des données sous forme de notification push.
- `sqs`pour envoyer des données vers une file d'attente Amazon SQS.

Utilisez le minuteur AWS IoT Events intégré et les actions variables

AWS IoT Events prend en charge les actions suivantes qui vous permettent d'utiliser un temporisateur ou de définir une variable :

- `setTimer`pour créer un chronomètre.
- `resetTimer`pour réinitialiser le chronomètre.
- `clearTimer`pour supprimer le chronomètre.
- `setVariable`pour créer une variable.

Régler l'action du chronomètre

Set timer action

L'`setTimer`action vous permet de créer un chronomètre dont la durée est exprimée en secondes.

More information (2)

Lorsque vous créez un temporisateur, vous devez spécifier les paramètres suivants.

`timerName`

Nom du minuteur.

`durationExpression`

(Facultatif) Durée du chronomètre, en secondes.

Le résultat évalué d'une expression de durée est arrondi au nombre entier inférieur le plus proche. Par exemple, si vous définissez le temporisateur sur 60,99 secondes, le résultat évalué de l'expression de durée est de 60 secondes.

Pour plus d’informations, consultez SetTimerAction dans la Référence d’API AWS IoT Events .

Action de réinitialisation du chronomètre

Reset timer action

L'`resetTimer`action vous permet de régler le chronomètre en fonction du résultat précédemment évalué de l'expression de durée.

More information (1)

Lorsque vous réinitialisez un temporisateur, vous devez spécifier le paramètre suivant.

`timerName`

Nom du minuteur.

AWS IoT Events ne réévalue pas l'expression de durée lorsque vous réinitialisez le chronomètre.

Pour plus d'informations, consultez ResetTimerAction dans la Référence d'API AWS IoT Events .

Effacer l'action du chronomètre

Clear timer action

L'`clearTimer`action vous permet de supprimer un chronomètre existant.

More information (1)

Lorsque vous supprimez un temporisateur, vous devez spécifier le paramètre suivant.

`timerName`

Nom du minuteur.

Pour plus d'informations, consultez ClearTimerAction dans la Référence d'API AWS IoT Events .

Définir une action variable

Set variable action

L'`setVariable`action vous permet de créer une variable avec une valeur spécifiée.

More information (2)

Lorsque vous créez une variable, vous devez spécifier les paramètres suivants.

`variableName`

Nom de la variable.

`value`

Nouvelle valeur de la variable.

Pour plus d'informations, consultez SetVariableAction dans la Référence d'API AWS IoT Events .

AWS IoT Events travailler avec d'autres AWS services

AWS IoT Events prend en charge les actions suivantes qui vous permettent de travailler avec AWS les services :

- `iotTopicPublish`pour publier un message sur un sujet MQTT.
- `iotEvents`pour envoyer des données en AWS IoT Events tant que valeur d'entrée.
- `iotSiteWise` pour envoyer des données à une propriété de ressources dans AWS IoT SiteWise.
- `dynamoDB`pour envoyer des données vers une table Amazon DynamoDB.
- `dynamoDBv2`pour envoyer des données vers une table Amazon DynamoDB.
- `firehose`pour envoyer des données vers un flux Amazon Data Firehose.
- `lambda`pour invoquer une AWS Lambda fonction.
- `sns`pour envoyer des données sous forme de notification push.
- `sqs`pour envoyer des données vers une file d'attente Amazon SQS.

> **Important**
>
> - Vous devez choisir la même AWS région pour les deux AWS IoT Events et pour les AWS services avec lesquels vous souhaitez travailler. Pour obtenir la liste des régions prises en charge, consultez Points de terminaison et quotas AWS IoT Events dans le Référence générale d'Amazon Web Services.

- Vous devez utiliser la même AWS région lorsque vous créez d'autres AWS ressources pour les AWS IoT Events actions. Si vous changez de AWS région, il se peut que vous rencontriez des problèmes pour accéder aux AWS ressources.

Par défaut, AWS IoT Events génère une charge utile standard au format JSON pour toute action. Cette charge utile d'action contient toutes les paires attribut-valeur contenant les informations relatives à l'instance du modèle de détecteur et à l'événement qui a déclenché l'action. Pour configurer la charge utile de l'action, vous pouvez utiliser une expression de contenu. Pour plus d'informations, consultez la section Expressions pour filtrer, transformer et traiter les données d'événements et le type de données de charge utile dans la référence de l'AWS IoT Events API.

AWS IoT Core

IoT topic publish action

L' AWS IoT Core action vous permet de publier un message MQTT via le courtier de AWS IoT messages. Pour obtenir la liste des régions prises en charge, consultez Points de terminaison et quotas AWS IoT Core dans le Référence générale d'Amazon Web Services.

Le courtier de AWS IoT messages connecte AWS IoT les clients en envoyant des messages des clients éditeurs aux clients abonnés. Pour plus d'informations, consultez la section Protocoles de communication des appareils dans le Guide du AWS IoT développeur.

More information (2)

Lorsque vous publiez un message MQTT, vous devez spécifier les paramètres suivants.

`mqttTopic`

Rubrique MQTT qui reçoit le message.

Vous pouvez définir un nom de rubrique MQTT de manière dynamique lors de l'exécution à l'aide de variables ou de valeurs d'entrée créées dans le modèle de détecteur.

`payload`

(Facultatif) La charge utile par défaut contient toutes les paires attribut-valeur contenant les informations relatives à l'instance du modèle de détecteur et à l'événement qui a déclenché l'action. Vous pouvez également personnaliser la charge utile. Pour plus d'informations, consultez la section Charge utile dans la référence de l'AWS IoT Events API.

> **Note**
>
> Assurez-vous que la politique associée à votre rôle AWS IoT Events de service accorde l'`iot:Publish`autorisation. Pour de plus amples informations, veuillez consulter Gestion des identités et des accès pour AWS IoT Events.

Pour plus d'informations, consultez IotTopicPublishAction dans la Référence d'API AWS IoT Events .

AWS IoT Events

IoT Events action

L' AWS IoT Events action vous permet d'envoyer des données en AWS IoT Events tant qu'entrée. Pour obtenir la liste des régions prises en charge, consultez Points de terminaison et quotas AWS IoT Events dans le Référence générale d'Amazon Web Services.

AWS IoT Events vous permet de surveiller votre parc d'équipements ou d'appareils pour détecter les pannes ou les changements de fonctionnement, et de déclencher des actions lorsque de tels événements se produisent. Pour plus d'informations, voir Qu'est-ce que c'est AWS IoT Events ? dans le Guide AWS IoT Events du développeur.

More information (2)

Lorsque vous envoyez des données à AWS IoT Events, vous devez spécifier les paramètres suivants.

`inputName`

Nom de l' AWS IoT Events entrée qui reçoit les données.

`payload`

(Facultatif) La charge utile par défaut contient toutes les paires attribut-valeur contenant les informations relatives à l'instance du modèle de détecteur et à l'événement qui a déclenché l'action. Vous pouvez également personnaliser la charge utile. Pour plus d'informations, consultez la section Charge utile dans la référence de l'AWS IoT Events API.

> **Note**
>
> Assurez-vous que la politique associée à votre rôle AWS IoT Events de service accorde l'`iotevents:BatchPutMessage`autorisation. Pour de plus amples informations, veuillez consulter Gestion des identités et des accès pour AWS IoT Events.

Pour plus d'informations, consultez IotEventsAction dans la Référence d'API AWS IoT Events .

AWS IoT SiteWise

IoT SiteWise action

L' AWS IoT SiteWise action vous permet d'envoyer des données à une propriété d'actif dans AWS IoT SiteWise. Pour obtenir la liste des régions prises en charge, consultez Points de terminaison et quotas AWS IoT SiteWise dans le Référence générale d'Amazon Web Services.

AWS IoT SiteWise est un service géré qui vous permet de collecter, d'organiser et d'analyser des données provenant d'équipements industriels à grande échelle. Pour plus d'informations, consultez Présentation de AWS IoT SiteWise dans le Guide de l'utilisateur AWS IoT SiteWise .

More information (11)

Lorsque vous envoyez des données à une propriété d'actif dans AWS IoT SiteWise, vous devez spécifier les paramètres suivants.

> **Important**
>
> Pour recevoir les données, vous devez utiliser une propriété d'actif existante dans AWS IoT SiteWise.
>
> - Si vous utilisez la AWS IoT Events console, vous devez spécifier `propertyAlias` pour identifier la propriété de l'actif cible.
> - Si vous utilisez le AWS CLI, vous devez spécifier l'un `propertyAlias` ou les deux `assetId` et `propertyId` identifier la propriété de l'actif cible.

Pour plus d'informations, veuillez consulter la rubrique Mappage de flux de données industrielles avec des propriétés de ressource dans le Guide de l'utilisateur AWS IoT SiteWise .

`propertyAlias`

(Facultatif) Alias de la propriété de l'actif. Vous pouvez également spécifier une expression.

`assetId`

(Facultatif) L'ID de l'actif qui possède la propriété spécifiée. Vous pouvez également spécifier une expression.

`propertyId`

(Facultatif) L'ID de la propriété de l'actif. Vous pouvez également spécifier une expression.

`entryId`

(Facultatif) Un identifiant unique pour cette entrée. Vous pouvez utiliser l'ID d'entrée pour suivre quelle entrée de données provoque une erreur en cas d'échec. La valeur par défaut est un nouvel identifiant unique. Vous pouvez également spécifier une expression.

`propertyValue`

Structure contenant des informations sur la valeur de la propriété.

`quality`

(Facultatif) La qualité de la valeur de la propriété de l'actif. La valeur doit être `GOOD`, `BAD` ou `UNCERTAIN`. Vous pouvez également spécifier une expression.

`timestamp`

(Facultatif) Structure contenant des informations d'horodatage. Si vous ne spécifiez pas cette valeur, la valeur par défaut est l'heure de l'événement.

`timeInSeconds`

Horodatage, en secondes, au format époque Unix. La plage valide est comprise entre 1 et 31556889864403199. Vous pouvez également spécifier une expression.

`offsetInNanos`

(Facultatif) Le décalage en nanosecondes converti à partir `detimeInSeconds`. La plage valide est comprise entre 0 et 999999999. Vous pouvez également spécifier une expression.

`value`

Structure contenant une valeur de propriété de ressource.

> **Important**
>
> Vous devez spécifier l'un des types de valeur suivants, en fonction du `dataType` de la propriété de ressource spécifiée. Pour plus d'informations, consultez AssetProperty dans la Référence d'API AWS IoT SiteWise .

`booleanValue`

(Facultatif) La valeur de la propriété de l'actif est une valeur booléenne qui doit être `TRUE` ou. `FALSE` Vous pouvez également spécifier une expression. Si vous utilisez une expression, le résultat évalué doit être une valeur booléenne.

`doubleValue`

(Facultatif) La valeur de la propriété de l'actif est un double. Vous pouvez également spécifier une expression. Si vous utilisez une expression, le résultat évalué doit être un double.

`integerValue`

(Facultatif) La valeur de la propriété de l'actif est un entier. Vous pouvez également spécifier une expression. Si vous utilisez une expression, le résultat évalué doit être un entier.

`stringValue`

(Facultatif) La valeur de la propriété de l'actif est une chaîne. Vous pouvez également spécifier une expression. Si vous utilisez une expression, le résultat évalué doit être une chaîne.

> **Note**
>
> Assurez-vous que la politique associée à votre rôle AWS IoT Events de service accorde l'`iotsitewise:BatchPutAssetPropertyValue`autorisation. Pour de plus amples informations, veuillez consulter Gestion des identités et des accès pour AWS IoT Events.

Pour plus d'informations, consultez IotSiteWiseAction dans la Référence d'API AWS IoT Events .

Amazon DynamoDB

DynamoDB action

L'action Amazon DynamoDB vous permet d'envoyer des données vers une table DynamoDB. Une colonne de la table DynamoDB reçoit toutes les paires attribut-valeur de la charge utile d'action que vous spécifiez. Pour obtenir la liste des régions prises en charge, consultez la section Points de terminaison et quotas Amazon DynamoDB dans le. Référence générale d'Amazon Web Services

Amazon DynamoDB est un service de base de données NoSQL entièrement géré, offrant des performances exceptionnelles et prévisibles en termes de rapidité et d'évolutivité. Pour plus d'informations, voir Qu'est-ce que DynamoDB ? dans le guide du développeur Amazon DynamoDB.

More information (10)

Lorsque vous envoyez des données vers une colonne d'une table DynamoDB, vous devez spécifier les paramètres suivants.

`tableName`

Nom de la table DynamoDB qui reçoit les données. La `tableName` valeur doit correspondre au nom de la table DynamoDB. Vous pouvez également spécifier une expression.

`hashKeyField`

Le nom de la clé de hachage (également appelée clé de partition). La `hashKeyField` valeur doit correspondre à la clé de partition de la table DynamoDB. Vous pouvez également spécifier une expression.

`hashKeyType`

(Facultatif) Type de données de la clé de hachage. La valeur du type de clé de hachage doit être `STRING` ou`NUMBER`. La valeur par défaut est `STRING`. Vous pouvez également spécifier une expression.

`hashKeyValue`

Valeur de la clé de hachage. Il `hashKeyValue` utilise des modèles de substitution. Ces modèles fournissent des données lors de l'exécution. Vous pouvez également spécifier une expression.

`rangeKeyField`

(Facultatif) Nom de la clé de plage (également appelée clé de tri). La `rangeKeyField` valeur doit correspondre à la clé de tri de la table DynamoDB. Vous pouvez également spécifier une expression.

`rangeKeyType`

(Facultatif) Type de données de la clé de plage. La valeur du type de clé de hachage doit être `STRING` ou`NUMBER`. La valeur par défaut est `STRING`. Vous pouvez également spécifier une expression.

`rangeKeyValue`

(Facultatif) Valeur de la clé de plage. Il `rangeKeyValue` utilise des modèles de substitution. Ces modèles fournissent des données lors de l'exécution. Vous pouvez également spécifier une expression.

fonctionnement

(Facultatif) Type d'opération à effectuer. Vous pouvez également spécifier une expression. La valeur de l'opération doit être l'une des valeurs suivantes :

- `INSERT` - Insérer des données en tant que nouvel élément dans la table DynamoDB. C’est la valeur par défaut.
- `UPDATE` - Mettre à jour un élément existant de la table DynamoDB avec de nouvelles données.
- `DELETE`- Supprime un élément existant de la table DynamoDB.

`payloadField`

(Facultatif) Nom de la colonne DynamoDB qui reçoit la charge utile de l'action. Le nom par défaut est `payload`. Vous pouvez également spécifier une expression.

`payload`

(Facultatif) La charge utile par défaut contient toutes les paires attribut-valeur contenant les informations relatives à l'instance du modèle de détecteur et à l'événement qui a déclenché l'action. Vous pouvez également personnaliser la charge utile. Pour plus d'informations, consultez la section Charge utile dans la référence de l'AWS IoT Events API.

Si le type de charge utile spécifié est une chaîne, `DynamoDBAction` envoie des données non JSON à la table DynamoDB sous forme de données binaires. La console DynamoDB affiche les données sous la forme de texte codé en Base64. La valeur `payloadField` est *`payload-field`*`_raw`. Vous pouvez également spécifier une expression.

> **Note**
>
> Assurez-vous que la politique associée à votre rôle AWS IoT Events de service accorde l'`dynamodb:PutItem`autorisation. Pour de plus amples informations, veuillez consulter Gestion des identités et des accès pour AWS IoT Events.

Pour plus d'informations, consultez Dynamo DBAction dans le manuel de référence des AWS IoT Events API.

Amazon DynamoDB (version 2)

DynamoDBv2 action

L'action Amazon DynamoDB (v2) vous permet d'écrire des données dans une table DynamoDB. Une colonne distincte de la table DynamoDB reçoit une paire attribut-valeur dans la charge utile d'action que vous spécifiez. Pour obtenir la liste des régions prises en charge, consultez la section Points de terminaison et quotas Amazon DynamoDB dans le. Référence générale d'Amazon Web Services

Amazon DynamoDB est un service de base de données NoSQL entièrement géré, offrant des performances exceptionnelles et prévisibles en termes de rapidité et d'évolutivité. Pour plus d'informations, voir Qu'est-ce que DynamoDB ? dans le guide du développeur Amazon DynamoDB.

More information (2)

Lorsque vous envoyez des données vers plusieurs colonnes d'une table DynamoDB, vous devez spécifier les paramètres suivants.

`tableName`

Nom de la table DynamoDB qui reçoit les données. Vous pouvez également spécifier une expression.

`payload`

(Facultatif) La charge utile par défaut contient toutes les paires attribut-valeur contenant les informations relatives à l'instance du modèle de détecteur et à l'événement qui a déclenché l'action. Vous pouvez également personnaliser la charge utile. Pour plus d'informations, consultez la section Charge utile dans la référence de l'AWS IoT Events API.

> **Important**
>
> Le type de charge utile doit être JSON. Vous pouvez également spécifier une expression.

> **Note**
>
> Assurez-vous que la politique associée à votre rôle AWS IoT Events de service accorde l'`dynamodb:PutItem`autorisation. Pour de plus amples informations, veuillez consulter Gestion des identités et des accès pour AWS IoT Events.

Pour plus d'informations, consultez Dynamo DBv2 Action dans le Guide de référence de l'AWS IoT Events API.

Amazon Data Firehose

Firehose action

L'action Amazon Data Firehose vous permet d'envoyer des données vers un flux de diffusion Firehose. Pour obtenir la liste des régions prises en charge, consultez la section Points de

terminaison et quotas Amazon Data Firehose dans le. Référence générale d'Amazon Web Services

Amazon Data Firehose est un service entièrement géré qui fournit des données de streaming en temps réel vers des destinations telles qu'Amazon Simple Storage Service (Amazon Simple Storage Service), Amazon Redshift, OpenSearch Amazon OpenSearch Service (Service) et Splunk. Pour plus d'informations, consultez Qu'est-ce qu'Amazon Data Firehose ? dans le manuel Amazon Data Firehose Developer Guide.

More information (3)

Lorsque vous envoyez des données à un flux de diffusion Firehose, vous devez spécifier les paramètres suivants.

`deliveryStreamName`

Nom du flux de diffusion Firehose qui reçoit les données.

`separator`

(Facultatif) Vous pouvez utiliser un séparateur de caractères pour séparer les données continues envoyées au flux de diffusion Firehose. La valeur du séparateur doit être `'\n'` (nouvelle ligne), `'\t'` (onglet), `'\r\n'` (nouvelle ligne Windows) ou `','` (virgule).

`payload`

(Facultatif) La charge utile par défaut contient toutes les paires attribut-valeur contenant les informations relatives à l'instance du modèle de détecteur et à l'événement qui a déclenché l'action. Vous pouvez également personnaliser la charge utile. Pour plus d'informations, consultez la section Charge utile dans la référence de l'AWS IoT Events API.

> **Note**
>
> Assurez-vous que la politique associée à votre rôle AWS IoT Events de service accorde l'`firehose:PutRecord`autorisation. Pour de plus amples informations, veuillez consulter Gestion des identités et des accès pour AWS IoT Events.

Pour plus d’informations, consultez FirehoseAction dans la Référence d’API AWS IoT Events .

AWS Lambda

Lambda action

L' AWS Lambda action vous permet d'appeler une fonction Lambda. Pour obtenir la liste des régions prises en charge, consultez Points de terminaison et quotas AWS Lambda dans le Référence générale d'Amazon Web Services.

AWS Lambda est un service de calcul qui vous permet d'exécuter du code sans provisionner ni gérer de serveurs. Pour plus d'informations, voir Qu'est-ce que c'est AWS Lambda ? dans le Guide AWS Lambda du développeur.

More information (2)

Lorsque vous appelez une fonction Lambda, vous devez spécifier les paramètres suivants.

`functionArn`

ARN de la fonction Lambda à appeler.

`payload`

(Facultatif) La charge utile par défaut contient toutes les paires attribut-valeur contenant les informations relatives à l'instance du modèle de détecteur et à l'événement qui a déclenché l'action. Vous pouvez également personnaliser la charge utile. Pour plus d'informations, consultez la section Charge utile dans la référence de l'AWS IoT Events API.

> **Note**
>
> Assurez-vous que la politique associée à votre rôle AWS IoT Events de service accorde l'`lambda:InvokeFunction`autorisation. Pour de plus amples informations, veuillez consulter Gestion des identités et des accès pour AWS IoT Events.

Pour plus d’informations, consultez LambdaAction dans la Référence d’API AWS IoT Events .

Amazon Simple Notification Service

SNS action

L'action de publication de rubriques Amazon SNS vous permet de publier un message Amazon SNS. Pour obtenir la liste des régions prises en charge, consultez la section Points de terminaison et quotas Amazon Simple Notification Service dans le Référence générale d'Amazon Web Services.

Amazon Simple Notification Service (Amazon Simple Notification Service) est un service Web qui coordonne et gère la livraison ou l'envoi de messages aux points de terminaison ou aux clients abonnés. Pour plus d'informations, consultez Qu'est-ce qu'Amazon SNS ? dans le guide du développeur d'Amazon Simple Notification Service.

> **Note**
>
> L'action de publication de rubriques Amazon SNS ne prend pas en charge les rubriques Amazon SNS FIFO (premier entré, premier sorti). Le moteur de règles étant un service entièrement distribué, les messages peuvent ne pas s'afficher dans un ordre spécifié lorsque l'action Amazon SNS est lancée.

More information (2)

Lorsque vous publiez un message Amazon SNS, vous devez spécifier les paramètres suivants.

`targetArn`

L'ARN de la cible Amazon SNS qui reçoit le message.

`payload`

(Facultatif) La charge utile par défaut contient toutes les paires attribut-valeur contenant les informations relatives à l'instance du modèle de détecteur et à l'événement qui a déclenché l'action. Vous pouvez également personnaliser la charge utile. Pour plus d'informations, consultez la section Charge utile dans la référence de l'AWS IoT Events API.

> **Note**
>
> Assurez-vous que la politique associée à votre rôle AWS IoT Events de service accorde l'`sns:Publish`autorisation. Pour de plus amples informations, veuillez consulter Gestion des identités et des accès pour AWS IoT Events.

Pour plus d'informations, consultez SNSTopicPublishAction dans la Référence d'API AWS IoT Events .

Amazon Simple Queue Service

SQS action

L'action Amazon SQS vous permet d'envoyer des données vers une file d'attente Amazon SQS. Pour obtenir la liste des régions prises en charge, consultez la section Points de terminaison et quotas Amazon Simple Queue Service dans le Référence générale d'Amazon Web Services.

Amazon Simple Queue Service (Amazon SQS) offre une file d'attente hébergée sécurisée, durable et disponible qui vous permet d'intégrer et de découpler les systèmes et les composants de logiciels distribués. Pour plus d'informations, consultez What is Amazon Simple Queue Service > dans le guide du développeur Amazon Simple Queue Service.

> **Note**
>
> L'action Amazon SQS ne prend pas en charge les rubriques >Amazon SQS FIFO (premier entré, premier sorti). Le moteur de règles étant un service entièrement distribué, les messages peuvent ne pas s'afficher dans un ordre spécifié lorsque l'action Amazon SQS est lancée.

More information (3)

Lorsque vous envoyez des données vers une file d'attente Amazon SQS, vous devez spécifier les paramètres suivants.

`queueUrl`

URL de la file d'attente Amazon SQS qui reçoit les données.

`useBase64`

(Facultatif) AWS IoT Events code les données en texte Base64, si vous le spécifiez. `TRUE` La valeur par défaut est `FALSE`.

`payload`

(Facultatif) La charge utile par défaut contient toutes les paires attribut-valeur contenant les informations relatives à l'instance du modèle de détecteur et à l'événement qui a déclenché l'action. Vous pouvez également personnaliser la charge utile. Pour plus d'informations, consultez la section Charge utile dans la référence de l'AWS IoT Events API.

> **Note**
>
> Assurez-vous que la politique associée à votre rôle AWS IoT Events de service accorde l'`sqs:SendMessage`autorisation. Pour de plus amples informations, veuillez consulter Gestion des identités et des accès pour AWS IoT Events.

Pour plus d'informations, consultez SNSTopicPublishAction dans la Référence d'API AWS IoT Events .

Vous pouvez également utiliser Amazon SNS et le moteur de AWS IoT Core règles pour déclencher une AWS Lambda fonction. Cela permet de prendre des mesures à l'aide d'autres services, tels qu'Amazon Connect, ou même d'une application de planification des ressources d'entreprise (ERP).

> **Note**
>
> Pour collecter et traiter de grands flux d'enregistrements de données en temps réel, vous pouvez utiliser d'autres AWS services, tels qu'Amazon Kinesis. À partir de là, vous pouvez effectuer une analyse initiale, puis envoyer les résultats AWS IoT Events sous forme d'entrée à un détecteur.

Expressions pour filtrer, transformer et traiter les données d'événements

Les expressions sont utilisées pour évaluer les données entrantes, effectuer des calculs et déterminer les conditions dans lesquelles des actions spécifiques ou des transitions d'état doivent se produire. AWS IoT Events propose plusieurs méthodes pour spécifier des valeurs lors de la création et de la mise à jour de modèles de détecteurs. Vous pouvez utiliser des expressions pour spécifier des valeurs littérales ou AWS IoT Events évaluer les expressions avant de spécifier des valeurs particulières.

Rubriques

- Syntaxe permettant de filtrer les données de l'appareil et de définir des actions dans AWS IoT Events
- Exemples d'expressions et utilisation pour AWS IoT Events

Syntaxe permettant de filtrer les données de l'appareil et de définir des actions dans AWS IoT Events

Les expressions proposent une syntaxe permettant de filtrer les données des appareils et de définir des actions. Vous pouvez utiliser des littéraux, des opérateurs, des fonctions, des références et des modèles de substitution dans les AWS IoT Events expressions. En combinant ces composants, vous pouvez créer des expressions puissantes et flexibles pour traiter les données IoT, effectuer des calculs, manipuler des chaînes et prendre des décisions logiques au sein de vos modèles de détecteurs.

Littéraux

- Entier
- Décimal
- String
- Booléen

Opérateurs

Unaire

- Non (booléen) : **!**
- Non (bit à bit) : **~**
- Moins (arithmétique) : **-**

String

- Concaténation : **+**

  Les deux opérandes doivent être des chaînes. Les chaînes littérales doivent être placées entre guillemets simples (').

  Par exemple : `'my' + 'string'` -> `'mystring'`

Arithmétique

- Ajout : **+**

  Les deux opérandes doivent être numériques.
- Soustraction : **-**
- Division : **/**

  Le résultat de la division est un entier arrondi sauf si au moins un des opérandes (diviseur ou dividende) est une valeur décimale.
- Multiplication : *****

Bit par bit (entier)

- OU : **|**

  Par exemple : `13 | 5` -> `13`
- ET : **&**

  Par exemple : `13 & 5` -> `5`
- PORTE : **^**

  Par exemple : `13 ^ 5` -> `8`
- NE PAS : **~**

  Par exemple : `~13` -> `-14`

Booléen

- Inférieur à : **<**
- Inférieur ou égal à : **<=**
- Égal à : **==**
- Non égal à : **!=**
- Supérieur ou égal à : **>=**
- Supérieur à : **>**
- ET : **&&**
- OU : **||**

> **Note**
>
> Lorsqu'une sous-expression de `||` contient des données non définies, cette sous-expression est traitée comme. `false`

Parenthèses

Vous pouvez utiliser des parenthèses pour regrouper les termes d'une expression.

Fonctions à utiliser dans les AWS IoT Events expressions

AWS IoT Events fournit un ensemble de fonctions intégrées pour améliorer les capacités des expressions de votre modèle de détecteur. Ces fonctions permettent la gestion du temporisateur, la conversion de type, la vérification nulle, l'identification du type de déclencheur, la vérification des entrées, la manipulation de chaînes et les opérations bit à bit. En tirant parti de ces fonctions, vous pouvez créer une logique de AWS IoT Events traitement réactive, améliorant ainsi l'efficacité globale de vos applications IoT.

Fonctions intégrées

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

Indique `true` si le délai spécifié est expiré. Remplacez « *`timer-name`* » par le nom d'un temporisateur que vous avez défini, entre guillemets. Dans le cadre d'une action événementielle, vous pouvez définir un chronomètre, puis le démarrer, le réinitialiser ou en effacer un que vous avez défini précédemment.

Voir le terraindetectorModelDefinition.states.onInput|onEnter|onExit.events.actions.setTimer.timerName.

Un temporisateur réglé dans un état peut être référencé dans un état différent. Vous devez consulter l'état dans lequel vous avez créé le chronomètre avant de saisir l'état dans lequel le chronomètre est référencé.

Par exemple, un modèle de détecteur possède deux états, TemperatureChecked etRecordUpdated. Vous avez créé un chronomètre dans l'TemperatureCheckedÉtat. Vous devez d'abord visiter l'TemperatureCheckedÉtat avant de pouvoir utiliser le chronomètre de l'RecordUpdatedÉtat.

Pour garantir la précision, la durée minimale pendant laquelle une minuterie doit être réglée est de 60 secondes.

> **Note**
>
> timeout()renvoie true uniquement la première fois qu'il est vérifié après l'expiration réelle du délai et revient false par la suite.

convert(*type*, *expression*)

Évalue la valeur de l'expression convertie dans le type spécifié. La *type* valeur doit être String, ouBoolean Decimal. Utilisez l'un de ces mots clés ou une expression qui correspond à une chaîne contenant le mot-clé. Seules les conversions suivantes réussissent et renvoient une valeur valide :

- Booléen -> chaîne

  Renvoie la chaîne "true" ou"false".
- Décimal -> chaîne
- Chaîne -> Booléen
- Chaîne -> décimal

  La chaîne spécifiée doit être une représentation valide d'un nombre décimal, sinon elle convert() échoue.

S'il convert() ne renvoie pas de valeur valide, l'expression dont il fait partie n'est pas non plus valide. Ce résultat est équivalent false et ne déclenchera pas le actions ou la

transition vers le paramètre `nextState` spécifié dans le cadre de l'événement au cours duquel l'expression apparaît.

`isNull`(*`expression`*)

Indique `true` si l'expression renvoie la valeur null. Par exemple, si l'entrée `MyInput` reçoit le message`{ "a": null }`, la valeur suivante est évaluée à`true`, mais est `isUndefined($input.MyInput.a)` évaluée à. `false`

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

`isUndefined`(*`expression`*)

Indique `true` si l'expression n'est pas définie. Par exemple, si l'entrée `MyInput` reçoit le message`{ "a": null }`, la valeur suivante est évaluée à`false`, mais est `isNull($input.MyInput.a)` évaluée à. `true`

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

`triggerType`("*`type`*")

La *`type`* valeur peut être `"Message"` ou`"Timer"`. Indique `true` si la condition d'événement dans laquelle il apparaît est en cours d'évaluation parce qu'un délai a expiré, comme dans l'exemple suivant.

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

Ou un message d'entrée a été reçu.

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

`currentInput`("*`input`*")

Indique `true` si la condition d'événement dans laquelle il apparaît est en cours d'évaluation parce que le message d'entrée spécifié a été reçu. Par exemple, si l'entrée `Command` reçoit le message`{ "value": "Abort" }`, la valeur suivante est évaluée à`true`.

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

Utilisez cette fonction pour vérifier que la condition est en cours d'évaluation parce qu'une entrée particulière a été reçue et qu'un délai n'a pas expiré, comme dans l'expression suivante.

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

Fonctions de correspondance de chaînes

`startsWith`(*`expression1`*, *`expression2`*)

Indique `true` si la première expression de chaîne commence par la deuxième expression de chaîne. Par exemple, si input `MyInput` reçoit le message`{ "status": "offline"}`, la valeur suivante est attribuée à`true`.

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

Les deux expressions doivent être évaluées à une valeur de chaîne. Si aucune des expressions n'est évaluée à une valeur de chaîne, le résultat de la fonction n'est pas défini. Aucune conversion n'est effectuée.

`endsWith`(*`expression1`*, *`expression2`*)

Indique `true` si la première expression sous forme de chaîne se termine par la deuxième expression sous forme de chaîne. Par exemple, si input `MyInput` reçoit le message`{ "status": "offline" }`, la valeur suivante est attribuée à`true`.

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

Les deux expressions doivent être évaluées à une valeur de chaîne. Si aucune des expressions n'est évaluée à une valeur de chaîne, le résultat de la fonction n'est pas défini. Aucune conversion n'est effectuée.

`contains`(*`expression1`*, *`expression2`*)

Indique `true` si la première expression de chaîne contient la deuxième expression de chaîne. Par exemple, si input `MyInput` reçoit le message`{ "status": "offline" }`, la valeur suivante est attribuée à`true`.

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

Les deux expressions doivent être évaluées à une valeur de chaîne. Si aucune des expressions n'est évaluée à une valeur de chaîne, le résultat de la fonction n'est pas défini. Aucune conversion n'est effectuée.

Fonctions de manipulation d'entiers au niveau du bit

bitor(*expression1*, *expression2*)

Évalue le OR au niveau du bit des expressions entières (l'opération binaire OR est effectuée sur les bits correspondants des entiers). Par exemple, si input `MyInput` reçoit le message`{ "value1": 13, "value2": 5 }`, la valeur suivante est attribuée à`13`.

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

Les deux expressions doivent être évaluées à une valeur entière. Si aucune des expressions n'est évaluée à une valeur entière, le résultat de la fonction n'est pas défini. Aucune conversion n'est effectuée.

bitand(*expression1*, *expression2*)

Évalue le ET par bit des expressions entières (l'opération binaire ET est effectuée sur les bits correspondants des entiers). Par exemple, si input `MyInput` reçoit le message`{ "value1": 13, "value2": 5 }`, la valeur suivante est attribuée à`5`.

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

Les deux expressions doivent être évaluées à une valeur entière. Si aucune des expressions n'est évaluée à une valeur entière, le résultat de la fonction n'est pas défini. Aucune conversion n'est effectuée.

bitxor(*expression1*, *expression2*)

Évalue le XOR bit à bit des expressions entières (l'opération XOR binaire est effectuée sur les bits correspondants des entiers). Par exemple, si input `MyInput` reçoit le message`{ "value1": 13, "value2": 5 }`, la valeur suivante est attribuée à`8`.

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

Les deux expressions doivent être évaluées à une valeur entière. Si aucune des expressions n'est évaluée à une valeur entière, le résultat de la fonction n'est pas défini. Aucune conversion n'est effectuée.

`bitnot(`*`expression`*`)`

Évalue le NOT par bit de l'expression entière (l'opération binaire NOT est effectuée sur les bits de l'entier). Par exemple, si input `MyInput` reçoit le message`{ "value": 13 }`, la valeur suivante est attribuée à`-14`.

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

Les deux expressions doivent être évaluées à une valeur entière. Si aucune des expressions n'est évaluée à une valeur entière, le résultat de la fonction n'est pas défini. Aucune conversion n'est effectuée.

AWS IoT Events référence pour les entrées et les variables dans les expressions

Inputs

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

`input-name`est une entrée que vous créez à l'aide de l'CreateInputaction.

Par exemple, si vous avez nommé une entrée `TemperatureInput` pour laquelle vous avez défini des `inputDefinition.attributes.jsonPath` entrées, les valeurs peuvent apparaître dans les champs disponibles suivants.

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

Pour référencer la valeur du `temperature` champ, utilisez la commande suivante.

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

Pour les champs dont les valeurs sont des tableaux, vous pouvez référencer les membres du tableau en utilisant`[`*`n`*`]`. Par exemple, étant donné les valeurs suivantes :

```
{
    "temperatures": [
```

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

La valeur `78.8` peut être référencée à l'aide de la commande suivante.

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

Variables

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

*`variable-name`*Il s'agit d'une variable que vous avez définie à l'aide de l'CreateDetectorModelaction.

Par exemple, si vous avez défini une variable nommée `TechnicianID` à l'aide de la commande suivante`detectorDefinition.states.onInputEvents.actions.setVariable.variableNam` vous pouvez référencer la dernière valeur (chaîne) donnée à la variable.

```
$variable.TechnicianID
```

Vous pouvez définir les valeurs des variables uniquement à l'aide de cette `setVariable` action. Vous ne pouvez pas attribuer de valeurs aux variables d'une expression. Une variable ne peut pas être désactivée. Par exemple, vous ne pouvez pas lui attribuer la valeur`null`.

> **Note**
>
> Dans les références qui utilisent des identifiants qui ne suivent pas le modèle (expression régulière)`[a-zA-Z][a-zA-Z0-9_]*`, vous devez placer ces identifiants dans des backticks (). ` Par exemple, une référence à une entrée nommée `MyInput` par un champ `_value` doit spécifier ce champ sous la forme`$input.MyInput.`_value``.

Lorsque vous utilisez des références dans des expressions, vérifiez les points suivants :

- Lorsque vous utilisez une référence comme opérande avec un ou plusieurs opérateurs, assurez-vous que tous les types de données auxquels vous faites référence sont compatibles.

Par exemple, dans l'expression suivante, le nombre entier 2 est un opérande des && opérateurs == et. Pour garantir la compatibilité des opérandes, `$variable.testVariable + 1` ils `$variable.testVariable` doivent faire référence à un entier ou à un nombre décimal.

De plus, le nombre entier 1 est un opérande de l'+opérateur. Par conséquent, `$variable.testVariable` doit faire référence à un entier ou à un nombre décimal.

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

- Lorsque vous utilisez une référence comme argument transmis à une fonction, assurez-vous que celle-ci prend en charge les types de données auxquels vous faites référence.

  Par exemple, la `timeout("`*`time-name`*`")` fonction suivante nécessite une chaîne avec des guillemets comme argument. Si vous utilisez une référence pour la *`timer-name`* valeur, vous devez référencer une chaîne entre guillemets.

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

  > **Note**
  >
  > Pour la `convert(`*`type`*`, `*`expression`*`)` fonction, si vous utilisez une référence pour la *`type`* valeur, le résultat évalué de votre référence doit être `StringDecimal`, ou`Boolean`.

AWS IoT Events les expressions prennent en charge les types de données entiers, décimaux, chaînes et booléens. Le tableau suivant fournit une liste de paires de types incompatibles.

Paires de types incompatibles
Entier, chaîne
Entier, booléen
Décimal, chaîne
Décimal, booléen
Chaîne, booléen

Modèles de substitution pour les AWS IoT Events expressions

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

`${}`Identifie la chaîne en tant que chaîne interpolée. Il `expression` peut s'agir de n'importe quelle AWS IoT Events expression. Cela inclut les opérateurs, les fonctions et les références.

Par exemple, vous avez utilisé cette SetVariableActionaction pour définir une variable. Le `variableName` est `SensorID` et l'`value` est `10`. Vous pouvez créer les modèles de substitution suivants.

Modèle de substitution	Chaîne de résultats
`'${'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}"`

Exemples d'expressions et utilisation pour AWS IoT Events

Vous pouvez spécifier des valeurs dans un modèle de détecteur de la manière suivante :

- Entrez les expressions prises en charge dans la AWS IoT Events console.
- Passez les expressions aux paramètres AWS IoT Events APIs as.

Les expressions prennent en charge les littéraux, les opérateurs, les fonctions, les références et les modèles de substitution.

> **Important**
>
> Vos expressions doivent faire référence à un entier, à un nombre décimal, à une chaîne ou à une valeur booléenne.

Écrire AWS IoT Events des expressions

Consultez les exemples suivants pour vous aider à écrire vos AWS IoT Events expressions :

Littéral

Pour les valeurs littérales, les expressions doivent contenir des guillemets simples. La valeur booléenne doit être soit`true`. `false`

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

Référence

Pour les références, vous devez spécifier des variables ou des valeurs d'entrée.

- L'entrée suivante fait référence à un nombre décimal,`10.01`.

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

- La variable suivante fait référence à une chaîne,`Greenhouse Temperature Table`.

  ```
  $variable.TableName
  ```

Modèle de substitution

Pour un modèle de substitution, vous devez utiliser `${}` et le modèle doit être entre guillemets simples. Un modèle de substitution peut également contenir une combinaison de littéraux, d'opérateurs, de fonctions, de références et de modèles de substitution.

- Le résultat évalué de l'expression suivante est une chaîne,`50.018 in Fahrenheit`.

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

- Le résultat évalué de l'expression suivante est une chaîne,`{\"sensor_id\": \"Sensor_1\",\"temperature\":\"50.018\"}`.

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

Concaténation de chaînes

Pour une concaténation de chaînes, vous devez utiliser +. Une concaténation de chaînes peut également contenir une combinaison de littéraux, d'opérateurs, de fonctions, de références et de modèles de substitution.

- Le résultat évalué de l'expression suivante est une chaîne,`Greenhouse Temperature Table 2000-01-01`.

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

AWS IoT Events exemples de modèles de détecteurs

Cette page fournit une liste d'exemples de cas d'utilisation qui montrent comment configurer différentes AWS IoT Events fonctionnalités. Les exemples vont des détections de base telles que les seuils de température aux scénarios plus avancés de détection d'anomalies et d'apprentissage automatique. Chaque exemple inclut des procédures et des extraits de code pour vous aider à configurer les AWS IoT Events détections, les actions et les intégrations. Ces exemples mettent en évidence la flexibilité du AWS IoT Events service et la manière dont il peut être personnalisé pour diverses applications et cas d'utilisation de l'IoT. Reportez-vous à cette page lorsque vous explorez AWS IoT Events les fonctionnalités ou si vous avez besoin de conseils pour implémenter un flux de travail de détection ou d'automatisation spécifique.

Rubriques

- Exemple : utilisation du contrôle de température HVAC avec AWS IoT Events
- Exemple : une grue détectant des conditions à l'aide de AWS IoT Events
- Envoyer des commandes en réponse à des conditions détectées dans AWS IoT Events
- Un modèle AWS IoT Events de détecteur pour la surveillance des grues
- AWS IoT Events entrées pour la surveillance des grues
- Envoyez des messages d'alarme et opérationnels avec AWS IoT Events
- Exemple : détection AWS IoT Events d'événements à l'aide de capteurs et d'applications
- Exemple : appareil avec HeartBeat lequel surveiller les connexions des appareils AWS IoT Events
- Exemple : une alarme ISA dans AWS IoT Events
- Exemple : créer une alarme simple avec AWS IoT Events

Exemple : utilisation du contrôle de température HVAC avec AWS IoT Events

Histoire de fond

Cet exemple implémente un modèle de régulation de température (un thermostat) doté des fonctionnalités suivantes :

- Vous définissez un modèle de détecteur capable de surveiller et de contrôler plusieurs zones. (Une instance de détecteur sera créée pour chaque zone.)

- Chaque instance de détecteur reçoit des données de température provenant de plusieurs capteurs placés dans chaque zone de contrôle.
- Vous pouvez modifier la température souhaitée (le point de réglage) pour chaque zone à tout moment.
- Vous pouvez définir les paramètres opérationnels pour chaque zone et modifier ces paramètres à tout moment.
- Vous pouvez ajouter ou supprimer des capteurs dans une zone à tout moment.
- Vous pouvez activer une durée de fonctionnement minimale pour les unités de chauffage et de refroidissement afin de les protéger contre les dommages.
- Les détecteurs rejetteront et signaleront les valeurs anormales des capteurs.
- Vous pouvez définir des points de consigne de température d'urgence. Si un capteur signale une température supérieure ou inférieure aux points de consigne que vous avez définis, les unités de chauffage ou de refroidissement seront immédiatement activées et le détecteur signalera ce pic de température.

Cet exemple illustre les fonctionnalités suivantes :

- Créez des modèles de détecteurs d'événements.
- Créez des entrées.
- Ingérez les entrées dans un modèle de détecteur.
- Évaluez les conditions de déclenchement.
- Reportez-vous aux variables d'état dans les conditions et définissez les valeurs des variables en fonction des conditions.
- Reportez-vous aux minuteries dans les conditions et réglez les minuteries en fonction des conditions.
- Effectuez des actions qui envoient des messages Amazon SNS et MQTT.

Définitions d'entrée pour un système CVC dans AWS IoT Events

A `seedTemperatureInput` est utilisé pour créer une instance de détecteur pour une zone et définir ses paramètres opérationnels.

La configuration des entrées des systèmes CVC AWS IoT Events est importante pour un contrôle climatique efficace. Cet exemple montre comment configurer des entrées qui capturent des

paramètres tels que les données de température, d'humidité, d'occupation et de consommation d'énergie. Apprenez à définir les attributs d'entrée, à configurer les sources de données et à configurer les règles de prétraitement pour aider vos modèles de détecteurs à recevoir des informations précises et opportunes pour une gestion et une efficacité optimales.

Commande CLI utilisée :

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

Dossier : `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" }
    ]
  }
}
```

Réponse :

```
{
    "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."
```

```
    }
}
```

Un `temperatureInput` doit être envoyé par chaque capteur dans chaque zone, si nécessaire.

Commande CLI utilisée :

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

Dossier : `temperatureInput.json`

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

Réponse :

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

Définition du modèle de détecteur pour un système CVC utilisant AWS IoT Events

`areaDetectorModel`définit le fonctionnement de chaque instance de détecteur. Chaque `state machine` instance ingère les relevés du capteur de température, puis change d'état et envoie des messages de contrôle en fonction de ces relevés.

Commande CLI utilisée :

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

Dossier : `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"
}
```

Réponse :

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

BatchPutMessage exemples pour un système CVC dans AWS IoT Events

Dans cet exemple, `BatchPutMessage` est utilisé pour créer une instance de détecteur pour une zone et définir les paramètres de fonctionnement initiaux.

Commande CLI utilisée :

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

Dossier : `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}"
    }
  ]
}
```

Réponse :

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

Dans cet exemple, `BatchPutMessage` il est utilisé pour signaler les relevés du capteur de température pour un seul capteur dans une zone.

Commande CLI utilisée :

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

Dossier : `temperatureExample.json`

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

```
  ]
}
```

Réponse :

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

Dans cet exemple, `BatchPutMessage` est utilisé pour modifier la température souhaitée pour une zone.

Commande CLI utilisée :

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

Dossier : `seedSetDesiredTemp.json`

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

Réponse :

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

Exemples de messages Amazon SNS générés par l'instance de `Area51` détection :

```
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"
}
```

Dans cet exemple, nous utilisons l'`DescribeDetector`API pour obtenir des informations sur l'état actuel d'une instance de détecteur.

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

Réponse :

```
{
    "detector": {
        "lastUpdateTime": 1557521572.216,
        "creationTime": 1557520274.405,
```

```
        "state": {
            "variables": [
                {
                    "name": "resetMe",
                    "value": "false"
                },
                {
                    "name": "rangeLow",
                    "value": "15.0"
                },
                {
                    "name": "noDelay",
                    "value": "false"
                },
                {
                    "name": "desiredTemperature",
                    "value": "20.0"
                },
                {
                    "name": "anomalousLow",
                    "value": "0.0"
                },
                {
                    "name": "sensorId",
                    "value": "\"01\""
                },
                {
                    "name": "sensorCount",
                    "value": "10"
                },
                {
                    "name": "rangeHigh",
                    "value": "30.0"
                },
                {
                    "name": "enteringNewState",
                    "value": "false"
                },
                {
                    "name": "averageTemperature",
                    "value": "19.572"
                },
                {
                    "name": "allowedError",
```

```
                    "value": "0.7"
                },
                {
                    "name": "anomalousHigh",
                    "value": "60.0"
                },
                {
                    "name": "reportedTemperature",
                    "value": "15.72"
                },
                {
                    "name": "goodToGo",
                    "value": "false"
                }
            ],
            "stateName": "idle",
            "timers": [
                {
                    "timestamp": 1557520454.0,
                    "name": "idleTimer"
                }
            ]
        },
        "keyValue": "Area51",
        "detectorModelName": "areaDetectorModel",
        "detectorModelVersion": "1"
    }
}
```

BatchUpdateDetector exemple pour un système CVC dans AWS IoT Events

Dans cet exemple, `BatchUpdateDetector` est utilisé pour modifier les paramètres opérationnels d'une instance de détecteur fonctionnelle.

La gestion efficace du système CVC nécessite souvent des mises à jour par lots de plusieurs détecteurs. Cette section explique comment utiliser la fonction AWS IoT Events de mise à jour par lots pour les détecteurs. Apprenez à modifier simultanément plusieurs paramètres de contrôle et à mettre à jour les valeurs de seuil afin de pouvoir ajuster les actions de réponse sur un parc d'appareils, améliorant ainsi votre capacité à gérer efficacement des systèmes à grande échelle.

Commande CLI utilisée :

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

Dossier : `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": [
        ]
      }
    }
  ]
}
```

Réponse :

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

Le moteur de AWS IoT Core règles et AWS IoT Events

Les règles suivantes republient les messages AWS IoT Events MQTT sous forme de messages de demande de mise à jour instantanée. Nous supposons que AWS IoT Core les éléments sont

définis pour une unité de chauffage et une unité de refroidissement pour chaque zone contrôlée par le modèle de détecteur.

Dans cet exemple, nous avons défini des éléments nommés `Area51HeatingUnit` et`Area51CoolingUnit`.

Commande CLI utilisée :

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

Dossier : `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"
        }
      }
    ]
  }
}
```

Réponse : [vide]

Commande CLI utilisée :

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

Dossier : `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"
        }
      }
    ]
  }
}
```

Réponse : [vide]

Commande CLI utilisée :

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

Dossier : `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"
        }
      }
    ]
```

```
  }
}
```

Réponse : [vide]

Commande CLI utilisée :

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

Dossier : `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"
        }
      }
    ]
  }
}
```

Réponse : [vide]

Exemple : une grue détectant des conditions à l'aide de AWS IoT Events

L'opérateur de nombreuses grues souhaite détecter le moment où les machines doivent être entretenues ou remplacées et déclencher les notifications appropriées. Chaque grue est équipée d'un moteur. Un moteur émet des messages (entrées) contenant des informations sur la pression et la température. L'opérateur souhaite deux niveaux de détecteurs d'événements :

- Un détecteur d'événements au niveau de la grue
- Un détecteur d'événements au niveau du moteur

À l'aide des messages provenant des moteurs (qui contiennent des métadonnées contenant à la fois le `craneId` et le`motorid`), l'opérateur peut exécuter les deux niveaux de détecteurs d'événements en utilisant un routage approprié. Lorsque les conditions de l'événement sont remplies, les notifications doivent être envoyées aux rubriques Amazon SNS appropriées. L'opérateur peut configurer les modèles de détecteurs de manière à ce que les notifications ne soient pas émises en double.

Cet exemple illustre les fonctionnalités suivantes :

- Création, lecture, mise à jour, suppression (CRUD) des entrées.
- Création, lecture, mise à jour, suppression (CRUD) de modèles de détecteurs d'événements et de différentes versions de détecteurs d'événements.
- Acheminement d'une entrée vers plusieurs détecteurs d'événements.
- Ingestion d'entrées dans un modèle de détecteur.
- Évaluation des conditions de déclenchement et des événements du cycle de vie.
- Possibilité de faire référence à des variables d'état dans des conditions et de définir leurs valeurs en fonction des conditions.
- Orchestration d'exécution avec définition, état, évaluateur de déclencheur et exécuteur d'actions.
- Exécution d'actions `ActionsExecutor` avec une cible SNS.

Envoyer des commandes en réponse à des conditions détectées dans AWS IoT Events

Cette page fournit un exemple d'utilisation de AWS IoT Events commandes pour configurer les entrées, créer des modèles de détecteurs et envoyer des données de capteurs simulées. Les exemples montrent comment tirer parti AWS IoT Events pour surveiller les équipements industriels tels que les moteurs et les grues.

```
#Create Pressure Input
aws iotevents create-input  --cli-input-json file://pressureInput.json
aws iotevents describe-input --input-name PressureInput
```

```
aws iotevents update-input  --cli-input-json file://pressureInput.json
aws iotevents list-inputs
aws iotevents delete-input --input-name PressureInput

#Create Temperature Input
aws iotevents create-input  --cli-input-json file://temperatureInput.json
aws iotevents describe-input --input-name TemperatureInput
aws iotevents update-input  --cli-input-json file://temperatureInput.json
aws iotevents list-inputs
aws iotevents delete-input --input-name TemperatureInput

#Create Motor Event Detector using pressure and temperature input
aws iotevents create-detector-model  --cli-input-json file://motorDetectorModel.json
aws iotevents describe-detector-model --detector-model-name motorDetectorModel
aws iotevents update-detector-model  --cli-input-json file://
updateMotorDetectorModel.json
aws iotevents list-detector-models
aws iotevents list-detector-model-versions --detector-model-name motorDetectorModel
aws iotevents delete-detector-model --detector-model-name motorDetectorModel

#Create Crane Event Detector using temperature input
aws iotevents create-detector-model  --cli-input-json file://craneDetectorModel.json
aws iotevents describe-detector-model --detector-model-name craneDetectorModel
aws iotevents update-detector-model  --cli-input-json file://
updateCraneDetectorModel.json
aws iotevents list-detector-models
aws iotevents list-detector-model-versions --detector-model-name craneDetectorModel
aws iotevents delete-detector-model --detector-model-name craneDetectorModel

#Replace craneIds
sed -i '' "s/100008/100009/g" messages/*

#Replace motorIds
sed -i '' "s/200008/200009/g" messages/*

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

#Send HighTemperature message
aws iotevents-data batch-put-message --cli-input-json file://messages/
highTemperatureMessage.json --cli-binary-format raw-in-base64-out

#Send LowPressure message
```

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

#Send LowTemperature message
aws iotevents-data batch-put-message --cli-input-json file://messages/
lowTemperatureMessage.json --cli-binary-format raw-in-base64-out
```

Un modèle AWS IoT Events de détecteur pour la surveillance des grues

Surveillez votre parc d'équipements ou d'appareils pour détecter les pannes ou les changements de fonctionnement, et déclenchez des actions lorsque de tels événements se produisent. Vous définissez des modèles de détecteurs au format JSON qui spécifient les états, les règles et les actions. Cela vous permet de surveiller les entrées telles que la température et la pression, de suivre les dépassements de seuils et d'envoyer des alertes. Les exemples montrent des modèles de détecteurs pour une grue et un moteur, détectant les problèmes de surchauffe et signalant par Amazon SNS lorsqu'un seuil est dépassé. Vous pouvez mettre à jour les modèles pour affiner le comportement sans perturber la surveillance.

Dossier : `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"
}
```

Pour mettre à jour un modèle de détecteur existant. Dossier : `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"
}
```

Dossier : 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"
}
```

Pour mettre à jour un modèle de détecteur existant. Dossier : `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 entrées pour la surveillance des grues

Dans cet exemple, nous montrons comment configurer les entrées pour un système de surveillance de grue à l'aide de AWS IoT Events. Il capture les entrées de pression et de température pour illustrer comment structurer les entrées pour la surveillance d'équipements industriels complexes.

Dossier : `pressureInput.json`

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

Dossier : `temperatureInput.json`

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

Envoyez des messages d'alarme et opérationnels avec AWS IoT Events

La gestion efficace des messages est importante dans les systèmes de surveillance des grues. Cette section explique comment configurer pour traiter et répondre AWS IoT Events à différents types de messages provenant des capteurs de grue. La configuration d'alarmes basées sur un message particulier peut vous aider à analyser, filtrer et acheminer les mises à jour de statut afin de déclencher les actions appropriées.

Dossier : `highPressureMessage.json`

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

        }
    ]
}
```

Dossier : highTemperatureMessage.json

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

Dossier : lowPressureMessage.json

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

Dossier : lowTemperatureMessage.json

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

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

Exemple : détection AWS IoT Events d'événements à l'aide de capteurs et d'applications

Ce modèle de détecteur est l'un des modèles disponibles sur la AWS IoT Events console. Il est inclus ici pour votre commodité.

Cet exemple illustre AWS IoT Events la détection des événements d'une application à l'aide de données de capteurs. Il montre comment créer un modèle de détecteur qui surveille des événements spécifiques afin de pouvoir déclencher les actions appropriées. Vous pouvez créer plusieurs entrées de capteurs, définir des conditions d'événement complexes et configurer des mécanismes de réponse gradués.

```
{
    "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"
    }
}
```

Exemple : appareil avec HeartBeat lequel surveiller les connexions des appareils AWS IoT Events

Ce modèle de détecteur est l'un des modèles disponibles sur la AWS IoT Events console. Il est inclus ici pour votre commodité.

L'exemple du rythme cardiaque défectueux (DHB) illustre comment il AWS IoT Events peut être utilisé dans le cadre de la surveillance des soins de santé. Cet exemple montre comment créer un modèle de détecteur qui analyse les données de fréquence cardiaque, détecte les modèles irréguliers et déclenche les réponses appropriées. Apprenez à configurer les entrées, à définir des seuils et à configurer des alertes en cas de problèmes cardiaques potentiels, en démontrant AWS IoT Events la polyvalence des applications de santé connexes.

```
{
```

```
    "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"
    }
}
```

Exemple : une alarme ISA dans AWS IoT Events

Ce modèle de détecteur est l'un des modèles disponibles sur la AWS IoT Events console. Il est inclus ici pour votre commodité.

```
{
    "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"
}
```

Exemple : créer une alarme simple avec AWS IoT Events

Ce modèle de détecteur est l'un des modèles disponibles sur la AWS IoT Events console. Il est inclus ici pour votre commodité.

```
{
    "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"
}
```

Surveillance avec alarmes intégrées AWS IoT Events

AWS IoT Events les alarmes vous aident à surveiller les modifications apportées à vos données. Les données peuvent être des indicateurs que vous mesurez pour votre équipement et vos processus. Vous pouvez créer des alarmes qui envoient des notifications lorsqu'un seuil est dépassé. Les alarmes vous aident à détecter les problèmes, à rationaliser la maintenance et à optimiser les performances de votre équipement et de vos processus.

Les alarmes sont des exemples de modèles d'alarme. Le modèle d'alarme indique ce qu'il faut détecter, quand envoyer des notifications, qui est averti, etc. Vous pouvez également spécifier une ou plusieurs actions prises en charge qui se produisent lorsque l'état de l'alarme change. AWS IoT Events achemine les attributs d'entrée dérivés de vos données vers les alarmes appropriées. Si les données que vous surveillez se situent en dehors de la plage spécifiée, l'alarme est déclenchée. Vous pouvez également accuser réception des alarmes ou les régler sur le mode snooze.

Travailler avec AWS IoT SiteWise

Vous pouvez utiliser des AWS IoT Events alarmes pour surveiller les propriétés des actifs dans AWS IoT SiteWise. AWS IoT SiteWise envoie les valeurs des propriétés des actifs aux AWS IoT Events alarmes. AWS IoT Events envoie l'état de l'alarme à AWS IoT SiteWise.

AWS IoT SiteWise prend également en charge les alarmes externes. Vous pouvez choisir des alarmes externes si vous utilisez des alarmes externes AWS IoT SiteWise et si vous disposez d'une solution qui renvoie les données d'état des alarmes. L'alarme externe contient une propriété de mesure qui ingère les données d'état de l'alarme.

AWS IoT SiteWise n'évalue pas l'état des alarmes externes. De plus, vous ne pouvez pas accuser réception ou suspendre une alarme externe lorsque l'état de l'alarme change.

Vous pouvez utiliser la fonction de SiteWise surveillance pour afficher l'état des alarmes externes dans les portails de SiteWise surveillance.

Pour plus d'informations, consultez les sections Surveillance des données à l'aide d'alarmes dans le guide de AWS IoT SiteWise l'utilisateur et Surveillance à l'aide d'alarmes dans le guide d'application du SiteWise moniteur.

Reconnaître le flux

Lorsque vous créez un modèle d'alarme, vous choisissez d'activer ou non le flux d'accusé de réception. Si vous activez le flux d'accusé de réception, votre équipe est avertie lorsque l'état de l'alarme change. Votre équipe peut accuser réception de l'alarme et laisser une note. Par exemple, vous pouvez inclure les informations relatives à l'alarme et les mesures que vous allez prendre pour résoudre le problème. Si les données que vous surveillez se situent en dehors de la plage spécifiée, l'alarme est déclenchée.

Les alarmes présentent les états suivants :

`DISABLED`

Lorsque l'alarme est activée, `DISABLED` elle n'est pas prête à évaluer les données. Pour activer l'alarme, vous devez la mettre en `NORMAL` état.

`NORMAL`

Lorsque l'alarme est activée, `NORMAL` elle est prête à évaluer les données.

`ACTIVE`

Si l'alarme est active`ACTIVE`, elle est invoquée. Les données que vous surveillez se situent en dehors de la plage spécifiée.

`ACKNOWLEDGED`

Lorsque l'alarme est activée, `ACKNOWLEDGED` elle a été déclenchée et vous avez accusé réception de l'alarme.

`LATCHED`

L'alarme a été déclenchée, mais vous n'avez pas accusé réception après un certain temps. L'alarme passe automatiquement à l'`NORMAL`état.

`SNOOZE_DISABLED`

Lorsque l'alarme est activée`SNOOZE_DISABLED`, elle est désactivée pendant une période spécifiée. Après le temps de rappel, l'alarme passe automatiquement à l'`NORMAL`état normal.

Création d'un modèle d'alarme dans AWS IoT Events

Vous pouvez utiliser des AWS IoT Events alarmes pour surveiller vos données et être averti lorsqu'un seuil est dépassé. Les alarmes fournissent des paramètres que vous pouvez utiliser pour créer ou

configurer un modèle d'alarme. Vous pouvez utiliser la AWS IoT Events console ou AWS IoT Events l'API pour créer ou configurer le modèle d'alarme. Lorsque vous configurez le modèle d'alarme, les modifications prennent effet à mesure que de nouvelles données arrivent.

Exigences

Les exigences suivantes s'appliquent lorsque vous créez un modèle d'alarme.

- Vous pouvez créer un modèle d'alarme pour surveiller un attribut d'entrée AWS IoT Events ou une propriété d'actif dans AWS IoT SiteWise.
  - Si vous choisissez de surveiller un attribut d'entrée dans AWS IoT Events, Créez une entrée pour les modèles dans AWS IoT Events avant de créer le modèle d'alarme.
  - Si vous choisissez de surveiller la propriété d'un actif, vous devez créer un modèle d'actif AWS IoT SiteWise avant de créer le modèle d'alarme.
- Vous devez disposer d'un rôle IAM qui permet à votre alarme d'effectuer des actions et d'accéder aux AWS ressources. Pour plus d'informations, consultez la section Configuration des autorisations pour AWS IoT Events.
- Toutes les AWS ressources utilisées dans ce didacticiel doivent se trouver dans la même AWS région.

Création d'un modèle d'alarme (console)

Ce qui suit explique comment créer un modèle d'alarme pour surveiller un AWS IoT Events attribut dans la AWS IoT Events console.

1. Connectez-vous à la console AWS IoT Events.
2. Dans le volet de navigation, sélectionnez Modèles d'alarme.
3. Sur la page Modèles d'alarme, choisissez Créer un modèle d'alarme.
4. Dans la section Détails du modèle d'alarme, procédez comme suit :
   a. Entrez un nom unique.
   b. (Facultatif) Entrez une description.
5. Dans la section Cible de l'alarme, procédez comme suit :

> **Important**
>
> Si vous choisissez la propriété de l'AWS IoT SiteWise actif, vous devez avoir créé un modèle d'actif dans AWS IoT SiteWise.

a. Choisissez l'attribut AWS IoT Events d'entrée.

b. Choisissez l'entrée.

c. Choisissez la clé d'attribut d'entrée. Cet attribut d'entrée est utilisé comme clé pour créer l'alarme. AWS IoT Events achemine les entrées associées à cette touche vers l'alarme.

> **Important**
>
> Si la charge utile du message d'entrée ne contient pas cette clé d'attribut d'entrée, ou si la clé ne se trouve pas dans le même chemin JSON spécifié dans la clé, l'ingestion du message échouera. AWS IoT Events

6. Dans la section Définitions des seuils, vous définissez l'attribut d'entrée, la valeur de seuil et l'opérateur de comparaison AWS IoT Events utilisés pour modifier l'état de l'alarme.

   a. Pour Attribut d'entrée, choisissez l'attribut que vous souhaitez surveiller.

      Chaque fois que cet attribut d'entrée reçoit de nouvelles données, il est évalué pour déterminer l'état de l'alarme.

   b. Pour Opérateur, choisissez l'opérateur de comparaison. L'opérateur compare votre attribut d'entrée à la valeur de seuil de votre attribut.

      Vous pouvez choisir l'une des options suivantes :

      - > supérieur à
      - >= supérieur ou égal à
      - < inférieur à
      - <= inférieur ou égal à
      - = égal à
      - ! = différent de

c. Pour la valeur du seuil, entrez un nombre ou choisissez un attribut dans les AWS IoT Events entrées. AWS IoT Events compare cette valeur avec la valeur de l'attribut d'entrée que vous avez choisi.

d. (Facultatif) Pour la gravité, utilisez un chiffre compris par votre équipe pour refléter la gravité de cette alarme.

7. (Facultatif) Dans la section Paramètres de notification, configurez les paramètres de notification pour l'alarme.

   Vous pouvez ajouter jusqu'à 10 notifications. Pour la notification 1, procédez comme suit :

   a. Pour Protocole, choisissez l'une des options suivantes :

      - Courrier électronique et texto - L'alarme envoie une notification par SMS et une notification par e-mail.
      - E-mail - L'alarme envoie une notification par e-mail.
      - Texte - L'alarme envoie une notification par SMS.

   b. Pour Expéditeur, spécifiez l'adresse e-mail qui peut envoyer des notifications concernant cette alarme.

      Pour ajouter d'autres adresses e-mail à votre liste d'expéditeurs, choisissez Ajouter un expéditeur.

   c. (Facultatif) Dans Destinataire, choisissez le destinataire.

      Pour ajouter d'autres utilisateurs à votre liste de destinataires, choisissez Ajouter un nouvel utilisateur. Vous devez ajouter de nouveaux utilisateurs à votre boutique IAM Identity Center avant de pouvoir les ajouter à votre modèle d'alarme. Pour de plus amples informations, veuillez consulter Gérez l'accès des destinataires des alarmes à l'IAM Identity Center dans AWS IoT Events.

   d. (Facultatif) Dans Message personnalisé supplémentaire, entrez un message qui décrit ce que l'alarme détecte et les mesures que les destinataires doivent prendre.

8. Dans la section Instance, vous pouvez activer ou désactiver toutes les instances d'alarme créées sur la base de ce modèle d'alarme.

9. Dans la section Paramètres avancés, procédez comme suit :

   a. Pour Accusez le flux, vous pouvez activer ou désactiver les notifications.

- Si vous choisissez Activé, vous recevez une notification lorsque l'état de l'alarme change. Vous devez accuser réception de la notification avant que l'état d'alarme ne redevienne normal.
- Si vous choisissez Désactivé, aucune action n'est requise. L'alarme passe automatiquement à l'état normal lorsque la mesure revient à la plage spécifiée.

Pour de plus amples informations, veuillez consulter Reconnaître le flux.

b. Pour Autorisations, choisissez l'une des options suivantes :

- Vous pouvez créer un nouveau rôle à partir de modèles de AWS politique et créer AWS IoT Events automatiquement un rôle IAM pour vous.
- Vous pouvez utiliser un rôle IAM existant qui permet à ce modèle d'alarme d'effectuer des actions et d'accéder à d'autres AWS ressources.

Pour plus d'informations, veuillez consulter la rubrique Gestion des identités et des accès pour AWS IoT Events.

c. Pour les paramètres de notification supplémentaires, vous pouvez modifier votre AWS Lambda fonction pour gérer les notifications d'alarme. Choisissez l'une des options suivantes pour votre AWS Lambda fonction :

- Créer une nouvelle AWS Lambda fonction : AWS IoT Events crée une nouvelle AWS Lambda fonction pour vous.
- Utiliser une AWS Lambda fonction existante : utilisez une AWS Lambda fonction existante en choisissant un nom de AWS Lambda fonction.

Pour plus d'informations sur les actions possibles, consultezAWS IoT Events travailler avec d'autres AWS services.

d. (Facultatif) Pour l'action Définir l'état, vous pouvez ajouter une ou plusieurs AWS IoT Events actions à effectuer lorsque l'état de l'alarme change.

10. (Facultatif) Vous pouvez ajouter des tags pour gérer vos alarmes. Pour plus d'informations, veuillez consulter la rubrique Balisage de vos AWS IoT Events ressources.
11. Choisissez Créer.

Répondre aux alarmes dans AWS IoT Events

Répondre efficacement aux alarmes est un aspect important de la gestion des systèmes IoT avec AWS IoT Events. Découvrez différentes manières de configurer et de gérer les alarmes, notamment : configurer des canaux de notification, définir des procédures d'escalade et mettre en œuvre des actions de réponse automatisées. Apprenez à créer des conditions d'alarme nuancées, à hiérarchiser les alertes et à les intégrer à d'autres AWS services afin de créer un système de gestion des alarmes réactif pour vos applications IoT.

Si vous avez activé le flux d'accusé de réception, vous recevez des notifications lorsque l'état de l'alarme change. Pour répondre à l'alarme, vous pouvez accuser réception, désactiver, activer, réinitialiser ou suspendre l'alarme.

Console

Voici comment réagir à une alarme dans la AWS IoT Events console.

1. Connectez-vous à la console AWS IoT Events.
2. Dans le volet de navigation, sélectionnez Modèles d'alarme.
3. Choisissez le modèle d'alarme cible.
4. Dans la section Liste des alarmes, choisissez l'alarme cible.
5. Vous pouvez choisir l'une des options suivantes dans Actions :

   - Confirmer - L'alarme passe à l'`ACKNOWLEDGED`état actuel.
   - Désactiver : l'alarme passe à l'`DISABLED`état normal.
   - Activer : l'alarme passe à l'`NORMAL`état actuel.
   - Réinitialisation : l'alarme passe à l'`NORMAL`état normal.
   - Snoooze, puis effectuez les opérations suivantes :

     1. Choisissez la durée du rappel ou entrez une durée de rappel personnalisée.
     2. Choisissez Enregistrer.

     L'alarme passe à l'`SNOOZE_DISABLED`état

   Pour plus d'informations sur les états des alarmes, consultezReconnaître le flux.

API

Pour répondre à une ou plusieurs alarmes, vous pouvez utiliser les opérations AWS IoT Events d'API suivantes :

- BatchAcknowledgeAlarm
- BatchDisableAlarm
- BatchEnableAlarm
- BatchResetAlarm
- BatchSnoozeAlarm

Gestion des notifications d'alarme dans AWS IoT Events

AWS IoT Events s'intègre à Lambda, offrant des fonctionnalités de traitement d'événements personnalisées. Cette section explique comment utiliser les fonctions Lambda dans vos modèles de AWS IoT Events détecteurs, ce qui vous permet d'exécuter une logique complexe, d'interagir avec des services externes et de mettre en œuvre une gestion sophistiquée des événements.

AWS IoT Events utilise une fonction Lambda pour gérer les notifications d'alarme. Vous pouvez utiliser la fonction Lambda fournie par AWS IoT Events ou en créer une nouvelle.

Rubriques

- Création d'une fonction Lambda dans AWS IoT Events
- À l'aide de la fonction Lambda fournie par AWS IoT Events
- Gérez l'accès des destinataires des alarmes à l'IAM Identity Center dans AWS IoT Events

Création d'une fonction Lambda dans AWS IoT Events

AWS IoT Events fournit une fonction Lambda qui permet aux alarmes d'envoyer et de recevoir des notifications par e-mail et SMS.

Exigences

Les exigences suivantes s'appliquent lorsque vous créez une fonction Lambda pour les alarmes :

- Si votre alarme envoie des notifications par SMS, assurez-vous qu'Amazon SNS est configuré pour envoyer des SMS.

- Pour plus d'informations, consultez la documentation suivante :
  - Messagerie texte mobile avec Amazon SNS et identités d'origine pour les messages SMS Amazon SNS dans le guide du développeur Amazon Simple Notification Service.
  - Qu'est-ce que la messagerie SMS destinée aux utilisateurs finaux d'AWS ? dans le guide de AWS SMS l'utilisateur.
- Si votre alarme envoie des notifications par e-mail ou SMS, vous devez disposer d'un rôle IAM qui vous permet AWS Lambda de travailler avec Amazon SES et Amazon SNS.

  Exemple de politique :

  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" : "*"
        }
    ]
}
```

- Vous devez choisir la même AWS région pour les deux AWS IoT Events et AWS Lambda. Pour la liste des régions prises en charge, voir AWS IoT Events points de terminaison et quotas et points de AWS Lambda terminaison et quotas dans le. Référence générale d'Amazon Web Services

Déployez une fonction Lambda pour utiliser AWS IoT Events CloudFormation

Ce didacticiel utilise un CloudFormation modèle pour déployer une fonction Lambda. Ce modèle crée automatiquement un rôle IAM qui permet à la fonction Lambda de fonctionner avec Amazon SES et Amazon SNS.

Ce qui suit vous montre comment utiliser le AWS Command Line Interface (AWS CLI) pour créer une CloudFormation pile.

1. Dans le terminal de votre appareil, exécutez `aws --version` pour vérifier si vous avez installé le AWS CLI. Pour plus d'informations, consultez la section Installation ou mise à jour vers la version la plus récente de l' AWS CLI dans le Guide de l'utilisateur AWS Command Line Interface .
2. Exécutez `aws configure list` pour vérifier si vous avez configuré le AWS CLI dans la AWS région qui contient toutes vos AWS ressources pour ce didacticiel. Pour plus d'informations, voir Définir et afficher les paramètres de configuration à l'aide des commandes du Guide de AWS Command Line Interface l'utilisateur
3. Téléchargez le CloudFormation modèle, NotificationLambda.Template.Yaml.zip.

> **Note**
>
> Si vous rencontrez des difficultés pour télécharger le fichier, le modèle est également disponible dans leCloudFormation modèle.

4. Décompressez le contenu et enregistrez-le localement en tant que `notificationLambda.template.yaml`.
5. Ouvrez un terminal sur votre appareil et accédez au répertoire dans lequel vous avez téléchargé le `notificationLambda.template.yaml` fichier.
6. Pour créer une CloudFormation pile, exécutez la commande suivante :

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

Vous pouvez modifier ce CloudFormation modèle pour personnaliser la fonction Lambda et son comportement.

> **Note**
>
> AWS Lambda réessaie deux fois les erreurs de fonction. Si la fonction ne dispose pas de la capacité suffisante pour gérer toutes les demandes entrantes, des événements peuvent attendre dans la file d'attente pendant des heures ou des jours avant d'être envoyés à la fonction. Vous pouvez configurer une file d'attente de messages non remis (DLQ) sur la fonction pour capturer les événements qui n'ont pas été traités correctement. Pour plus d'informations, consultez Appel asynchrone dans le Guide du développeur AWS Lambda .

Vous pouvez également créer ou configurer la pile dans la CloudFormation console. Pour plus d'informations, consultez la section Utilisation des piles dans le Guide de l'AWS CloudFormation utilisateur.

Création d'une fonction Lambda personnalisée pour AWS IoT Events

Vous pouvez créer une fonction Lambda ou modifier celle fournie par. AWS IoT Events

Les exigences suivantes s'appliquent lorsque vous créez une fonction Lambda personnalisée.

- Ajoutez des autorisations qui permettent à votre fonction Lambda d'effectuer des actions spécifiques et d'accéder aux AWS ressources.
- Si vous utilisez la fonction Lambda fournie par AWS IoT Events, assurez-vous de choisir le runtime Python 3.7.

Exemple de fonction Lambda :

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

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

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

Pour plus d'informations, veuillez consulter Présentation de AWS Lambda dans le Manuel du développeur AWS Lambda .

CloudFormation modèle

Utilisez le CloudFormation modèle suivant pour créer votre fonction Lambda.

```
AWSTemplateFormatVersion: '2010-09-09'
Description: 'Notification Lambda for Alarm Model'
Resources:
  NotificationLambdaRole:
    Type: AWS::IAM::Role
    Properties:
      AssumeRolePolicyDocument:
        Statement:
          - Effect: Allow
            Principal:
              Service: lambda.amazonaws.com
            Action: sts:AssumeRole
      Path: "/"
      ManagedPolicyArns:
        - 'arn:aws:iam::aws:policy/AWSLambdaExecute'
      Policies:
```

```
        - PolicyName: "NotificationLambda"
          PolicyDocument:
            Version: "2012-10-17"
            Statement:
              - Effect: "Allow"
                Action:
                  - "ses:GetIdentityVerificationAttributes"
                  - "ses:SendEmail"
                  - "ses:VerifyEmailIdentity"
                Resource: "*"
              - Effect: "Allow"
                Action:
                  - "sns:Publish"
                  - "sns:OptInPhoneNumber"
                  - "sns:CheckIfPhoneNumberIsOptedOut"
                  - "sms-voice:DescribeOptedOutNumbers"
                Resource: "*"
              - Effect: "Deny"
                Action:
                  - "sns:Publish"
                Resource: "arn:aws:sns:*:*:*"
  NotificationLambdaFunction:
    Type: AWS::Lambda::Function
    Properties:
      Role: !GetAtt NotificationLambdaRole.Arn
      Runtime: python3.7
      Handler: index.lambda_handler
      Timeout: 300
      MemorySize: 3008
      Code:
        ZipFile: |
          import boto3
          import json
          import logging
          import datetime
          logger = logging.getLogger()
          logger.setLevel(logging.INFO)
          ses = boto3.client('ses')
          sns = boto3.client('sns')
          def check_value(target):
            if target:
              return True
            return False
```

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

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

À l'aide de la fonction Lambda fournie par AWS IoT Events

Avec les notifications d'alarme, vous pouvez utiliser la fonction Lambda fournie par AWS IoT Events pour gérer les notifications d'alarme.

Les exigences suivantes s'appliquent lorsque vous utilisez la fonction Lambda fournie par AWS IoT Events pour gérer vos notifications d'alarme :

- Vous devez vérifier l'adresse e-mail qui envoie les notifications par e-mail dans Amazon Simple Email Service (Amazon SES). Pour plus d'informations, consultez Vérifier l'identité d'une adresse e-mail dans le manuel Amazon Simple Email Service Developer Guide.

  Si vous recevez un lien de vérification, cliquez dessus pour vérifier votre adresse e-mail. Vous pouvez également vérifier la présence d'un e-mail de vérification dans votre dossier de courrier indésirable.
- Si votre alarme envoie des notifications par SMS, vous devez utiliser le format de numéro de téléphone international E.164 pour les numéros de téléphone. Ce format contient+`<country-calling-code><area-code><phone-number>`.

  Exemples de numéros de téléphone :

Country	Numéro de téléphone local	Numéro au format E.164
États-Unis	206-555-0100	+12065550100
Royaume-Uni	020-1234-1234	+442012341234
Lituanie	8+601+12345	+37060112345

  Pour trouver le code d'appel d'un pays, rendez-vous sur countrycode.org.

  La fonction Lambda fournie par AWS IoT Events vérifie si vous utilisez des numéros de téléphone au format E.164. Cependant, il ne vérifie pas les numéros de téléphone. Si vous vous assurez que vous avez saisi des numéros de téléphone exacts mais que vous n'avez pas reçu de notifications

par SMS, vous pouvez contacter les opérateurs téléphoniques. Les transporteurs peuvent bloquer les messages.

Gérez l'accès des destinataires des alarmes à l'IAM Identity Center dans AWS IoT Events

AWS IoT Events permet AWS IAM Identity Center de gérer l'accès SSO des destinataires des alarmes. La mise en œuvre d'IAM Identity Center pour les destinataires des AWS IoT Events notifications peut améliorer la sécurité et l'expérience utilisateur. Pour permettre à l'alarme d'envoyer des notifications aux destinataires, vous devez activer IAM Identity Center et ajouter des destinataires à votre boutique IAM Identity Center. Pour plus d'informations, consultez la section Ajouter des utilisateurs dans le guide de AWS IAM Identity Center l'utilisateur.

⚠ Important

- Vous devez choisir la même AWS région pour AWS IoT Events AWS Lambda, et le centre d'identité IAM.
- AWS Organizations ne prend en charge qu'une seule région du centre d'identité IAM à la fois. Si vous souhaitez rendre IAM Identity Center disponible dans une autre région, vous devez d'abord supprimer votre configuration IAM Identity Center actuelle. Pour plus d'informations, consultez les données régionales du centre d'identité IAM dans le guide de AWS IAM Identity Center l'utilisateur.

Sécurité dans AWS IoT Events

La sécurité du cloud AWS est la priorité absolue. En tant que AWS client, vous bénéficiez d'un centre de données et d'une architecture réseau conçus pour répondre aux exigences des entreprises les plus sensibles en matière de sécurité.

La sécurité est une responsabilité partagée entre vous AWS et vous. Le modèle de responsabilité partagée décrit cette notion par les termes sécurité du cloud et sécurité dans le cloud :

- Sécurité du cloud : AWS est chargée de protéger l'infrastructure qui exécute les AWS services dans le AWS cloud. AWS vous fournit également des services que vous pouvez utiliser en toute sécurité. L'efficacité de notre sécurité est régulièrement testée et vérifiée par des auditeurs tiers dans le cadre des programmes de conformitéAWS. Pour en savoir plus sur les programmes de conformité applicables AWS IoT Events, consultez la section AWS Services concernés par programme de conformité.
- Sécurité dans le cloud — Votre responsabilité est déterminée par le AWS service que vous utilisez. Vous êtes également responsable d'autres facteurs, y compris la sensibilité de vos données, les exigences de votre organisation, et la législation et la réglementation applicables.

Cette documentation vous aidera à comprendre comment appliquer le modèle de responsabilité partagée lors de son utilisation AWS IoT Events. Les rubriques suivantes expliquent comment procéder à la configuration AWS IoT Events pour atteindre vos objectifs de sécurité et de conformité. Vous apprendrez également à utiliser d'autres AWS services qui peuvent vous aider à surveiller et à sécuriser vos AWS IoT Events ressources.

Rubriques

- Gestion des identités et des accès pour AWS IoT Events
- Surveillance AWS IoT Events pour maintenir la fiabilité, la disponibilité et les performances
- Validation de conformité pour AWS IoT Events
- Résilience dans AWS IoT Events
- Sécurité de l'infrastructure dans AWS IoT Events

Gestion des identités et des accès pour AWS IoT Events

Gestion des identités et des accès AWS (IAM) est un AWS service qui aide un administrateur à contrôler en toute sécurité l'accès aux AWS ressources. Les administrateurs IAM contrôlent qui peut être authentifié (connecté) et autorisé (autorisé) à utiliser AWS IoT Events les ressources. IAM est un AWS service que vous pouvez utiliser sans frais supplémentaires.

Rubriques

- Public ciblé
- Authentification par des identités
- Gestion de l’accès à l’aide de politiques
- En savoir plus sur la gestion des identités et des accès
- Comment AWS IoT Events fonctionne avec IAM
- AWS IoT Events exemples de politiques basées sur l'identité
- Prévention interservices confuse des adjoints pour AWS IoT Events
- Résoudre les problèmes d' AWS IoT Events identité et d'accès

Public ciblé

La façon dont vous utilisez Gestion des identités et des accès AWS (IAM) varie en fonction de votre rôle :

- Utilisateur du service : demandez des autorisations à votre administrateur si vous ne pouvez pas accéder aux fonctionnalités (voir Résoudre les problèmes d' AWS IoT Events identité et d'accès)
- Administrateur du service : déterminez l’accès des utilisateurs et soumettez les demandes d’autorisation (voir Comment AWS IoT Events fonctionne avec IAM)
- Administrateur IAM : rédigez des politiques pour gérer l’accès (voir AWS IoT Events exemples de politiques basées sur l'identité)

Authentification par des identités

L'authentification est la façon dont vous vous connectez à AWS l'aide de vos informations d'identification. Vous devez être authentifié en tant qu'utilisateur IAM ou en assumant un rôle IAM. Utilisateur racine d'un compte AWS

Vous pouvez vous connecter en tant qu'identité fédérée à l'aide d'informations d'identification provenant d'une source d'identité telle que AWS IAM Identity Center (IAM Identity Center), d'une authentification unique ou d'informations d'identification. Google/Facebook Pour plus d'informations sur la connexion, consultez Connexion à votre Compte AWS dans le Guide de l'utilisateur Connexion à AWS .

Pour l'accès par programmation, AWS fournit un SDK et une CLI pour signer les demandes de manière cryptographique. Pour plus d'informations, consultez Signature AWS Version 4 pour les demandes d'API dans le Guide de l'utilisateur IAM.

Compte AWS utilisateur root

Lorsque vous créez un Compte AWS, vous commencez par une seule identité de connexion appelée utilisateur Compte AWS root qui dispose d'un accès complet à toutes Services AWS les ressources. Il est vivement déconseillé d'utiliser l'utilisateur racine pour vos tâches quotidiennes. Pour les tâches qui requièrent des informations d'identification de l'utilisateur racine, consultez Tâches qui requièrent les informations d'identification de l'utilisateur racine dans le Guide de l'utilisateur IAM.

Utilisateurs et groupes IAM

Un utilisateur IAM est une identité qui dispose d'autorisations spécifiques pour une seule personne ou application. Nous vous recommandons d'utiliser ces informations d'identification temporaires au lieu des utilisateurs IAM avec des informations d'identification à long terme. Pour plus d'informations, voir Exiger des utilisateurs humains qu'ils utilisent la fédération avec un fournisseur d'identité pour accéder à AWS l'aide d'informations d'identification temporaires dans le guide de l'utilisateur IAM.

Les groupes IAM spécifient une collection d'utilisateurs IAM et permettent de gérer plus facilement les autorisations pour de grands ensembles d'utilisateurs. Pour plus d'informations, consultez Cas d'utilisation pour les utilisateurs IAM dans le Guide de l'utilisateur IAM.

Rôles IAM

Un rôle IAM est une identité dotée d'autorisations spécifiques qui fournit des informations d'identification temporaires. Vous pouvez assumer un rôle en passant d'un rôle utilisateur à un rôle IAM (console) ou en appelant une opération AWS CLI ou AWS API. Pour plus d'informations, consultez Méthodes pour endosser un rôle dans le Guide de l'utilisateur IAM.

Les rôles IAM sont utiles pour l'accès des utilisateurs fédérés, les autorisations temporaires des utilisateurs IAM, les accès intercompte, les accès entre services et les applications exécutées sur

Amazon EC2. Pour plus d'informations, consultez Accès intercompte aux ressources dans IAM dans le Guide de l'utilisateur IAM.

Gestion de l'accès à l'aide de politiques

Vous contrôlez l'accès en AWS créant des politiques et en les associant à AWS des identités ou à des ressources. Une politique définit les autorisations lorsqu'elles sont associées à une identité ou à une ressource. AWS évalue ces politiques lorsqu'un directeur fait une demande. La plupart des politiques sont stockées AWS sous forme de documents JSON. Pour plus d'informations les documents de politique JSON, consultez Vue d'ensemble des politiques JSON dans le Guide de l'utilisateur IAM.

À l'aide de politiques, les administrateurs précisent qui a accès à quoi en définissant quel principal peut effectuer des actions sur quelles ressources et dans quelles conditions.

Par défaut, les utilisateurs et les rôles ne disposent d'aucune autorisation. Un administrateur IAM crée des politiques IAM et les ajoute aux rôles, que les utilisateurs peuvent ensuite assumer. Les politiques IAM définissent les autorisations quelle que soit la méthode que vous utilisez pour exécuter l'opération.

Politiques basées sur l'identité

Les stratégies basées sur l'identité sont des documents de stratégie d'autorisations JSON que vous attachez à une identité (utilisateur, groupe ou rôle). Ces politiques contrôlent les actions que peuvent exécuter ces identités, sur quelles ressources et dans quelles conditions. Pour découvrir comment créer une politique basée sur l'identité, consultez Définition d'autorisations IAM personnalisées avec des politiques gérées par le client dans le Guide de l'utilisateur IAM.

Les politiques basées sur l'identité peuvent être des politiques intégrées (intégrées directement dans une seule identité) ou des politiques gérées (politiques autonomes associées à plusieurs identités). Pour découvrir comment choisir entre des politiques gérées et en ligne, consultez Choix entre les politiques gérées et les politiques en ligne dans le Guide de l'utilisateur IAM.

Autres types de politique

AWS prend en charge des types de politiques supplémentaires qui peuvent définir les autorisations maximales accordées par les types de politiques les plus courants :

- Limites d'autorisations : une limite des autorisations définit le nombre maximum d'autorisations qu'une politique basée sur l'identité peut accorder à une entité IAM. Pour plus d'informations, consultez Limites d'autorisations pour des entités IAM dans le Guide de l'utilisateur IAM.
- Politiques de contrôle des services (SCPs) — Spécifiez les autorisations maximales pour une organisation ou une unité organisationnelle dans AWS Organizations. Pour plus d'informations, consultez Politiques de contrôle de service dans le Guide de l'utilisateur AWS Organizations .
- Politiques de contrôle des ressources (RCPs) : définissez le maximum d'autorisations disponibles pour les ressources de vos comptes. Pour plus d'informations, voir Politiques de contrôle des ressources (RCPs) dans le guide de AWS Organizations l'utilisateur.
- Politiques de session : politiques avancées que vous passez en tant que paramètre lorsque vous créez par programmation une session temporaire pour un rôle ou un utilisateur fédéré. Pour plus d'informations, consultez Politiques de session dans le Guide de l'utilisateur IAM.

Plusieurs types de politique

Lorsque plusieurs types de politiques s'appliquent à la requête, les autorisations en résultant sont plus compliquées à comprendre. Pour savoir comment AWS déterminer s'il faut autoriser une demande lorsque plusieurs types de politiques sont impliqués, consultez la section Logique d'évaluation des politiques dans le guide de l'utilisateur IAM.

En savoir plus sur la gestion des identités et des accès

Pour plus d'informations sur la gestion des identités et des accès pour AWS IoT Events, consultez les pages suivantes :

- Comment AWS IoT Events fonctionne avec IAM
- Résoudre les problèmes d' AWS IoT Events identité et d'accès

Comment AWS IoT Events fonctionne avec IAM

Avant d'utiliser IAM pour gérer l'accès à AWS IoT Events, vous devez connaître les fonctionnalités IAM disponibles. AWS IoT Events Pour obtenir une vue d'ensemble de la façon dont AWS IoT Events les autres AWS services fonctionnent avec IAM, consultez la section AWS Services compatibles avec IAM dans le Guide de l'utilisateur d'IAM.

Rubriques

- AWS IoT Events politiques basées sur l'identité
- AWS IoT Events politiques basées sur les ressources
- Autorisation basée sur les AWS IoT Events tags
- AWS IoT Events Rôles IAM

AWS IoT Events politiques basées sur l'identité

Avec les politiques basées sur l'identité IAM, vous pouvez spécifier des actions et ressources autorisées ou refusées, ainsi que les conditions dans lesquelles les actions sont autorisées ou refusées. AWS IoT Events prend en charge des actions, ressources et clés de condition spécifiques. Pour en savoir plus sur tous les éléments que vous utilisez dans une politique JSON, consultez Références des éléments de politique JSON IAM dans le Guide de l'utilisateur IAM.

Actions

L'élément `Action` d'une stratégie basée sur une identité IAM décrit les actions spécifiques qui seront autorisées ou refusées par la stratégie. Les actions de stratégie portent généralement le même nom que l'opération AWS d'API associée. L'action est utilisée dans une politique pour permettre d'effectuer l'opération associée.

Les actions de politique en AWS IoT Events cours utilisent le préfixe suivant avant l'action :`iotevents:`. Par exemple, pour autoriser quelqu'un à créer une AWS IoT Events entrée avec l'opération d' AWS IoT Events `CreateInput`API, vous devez inclure l'`iotevents:CreateInput`action dans sa politique. Pour autoriser quelqu'un à envoyer une entrée avec l'opération d' AWS IoT Events `BatchPutMessage`API, vous devez inclure l'`iotevents-data:BatchPutMessage`action dans sa politique. Les déclarations de politique doivent inclure un `NotAction` élément `Action` ou. AWS IoT Events définit son propre ensemble d'actions décrivant les tâches que vous pouvez effectuer avec ce service.

Pour spécifier plusieurs actions dans une seule déclaration, séparez-les par des virgules comme suit :

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

Vous pouvez aussi spécifier plusieurs actions à l’aide de caractères génériques (*). Par exemple, pour spécifier toutes les actions qui commencent par le mot `Describe`, incluez l’action suivante :

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

Pour consulter la liste des AWS IoT Events actions, reportez-vous à la section Actions définies par AWS IoT Events dans le guide de l'utilisateur IAM.

Ressources

L'élément `Resource` précise les objets auxquels l'action s'applique. Les instructions doivent inclure un élément `Resource` ou `NotResource`. Vous spécifiez une ressource à l'aide d'un ARN ou du caractère générique (*) pour indiquer que l'instruction s'applique à toutes les ressources.

La ressource du modèle de AWS IoT Events détecteur possède l'ARN suivant :

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

Pour plus d'informations sur le format de ARNs, consultez Identifier les AWS ressources avec Amazon Resource Names (ARNs).

Par exemple, pour spécifier le modèle du `Foobar` détecteur dans votre instruction, utilisez l'ARN suivant :

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

Pour spécifier toutes les instances qui appartiennent à un compte spécifique, utilisez le caractère générique (*) :

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

Certaines AWS IoT Events actions, telles que celles relatives à la création de ressources, ne peuvent pas être effectuées sur une ressource spécifique. Dans ces cas-là, vous devez utiliser le caractère générique (*).

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

Certaines actions AWS IoT Events d'API impliquent plusieurs ressources. Par exemple, `CreateDetectorModel` fait référence aux entrées dans ses instructions de condition, de sorte

qu'un utilisateur doit être autorisé à utiliser l'entrée et le modèle de détecteur. Pour spécifier plusieurs ressources dans une seule instruction, séparez-les ARNs par des virgules.

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

Pour consulter la liste des types de AWS IoT Events ressources et leurs caractéristiques ARNs, reportez-vous à la section Ressources définies par AWS IoT Events dans le guide de l'utilisateur IAM. Pour savoir grâce à quelles actions vous pouvez spécifier l'ARN de chaque ressource, consultez Actions définies par AWS IoT Events.

Clés de condition

L'élément `Condition` (ou le bloc `Condition`) vous permet de spécifier des conditions lorsqu'une instruction est appliquée. L'élément `Condition` est facultatif. Vous pouvez créer des expressions conditionnelles qui utilisent des opérateurs de condition, comme égal ou inférieur, pour faire correspondre la condition de la stratégie aux valeurs de la demande.

Si vous spécifiez plusieurs éléments `Condition` dans une instruction, ou plusieurs clés dans un seul élément `Condition`, AWS les évalue à l'aide d'une opération `AND` logique. Si vous spécifiez plusieurs valeurs pour une seule clé de condition, AWS évalue la condition à l'aide d'une `OR` opération logique. Toutes les conditions doivent être remplies avant que les autorisations associées à l'instruction ne soient accordées.

Vous pouvez aussi utiliser des variables d'espace réservé quand vous spécifiez des conditions. Par exemple, vous pouvez accorder à un utilisateur l'autorisation d'accéder à une ressource uniquement si elle est balisée avec son nom d'utilisateur. Pour plus d'informations, consultez Éléments des politiques IAM : variables et balises dans le Guide de l'utilisateur IAM.

AWS IoT Events ne fournit aucune clé de condition spécifique au service, mais il prend en charge l'utilisation de certaines clés de condition globales. Pour voir toutes les clés de condition AWS globales, voir les clés de contexte de condition AWS globales dans le guide de l'utilisateur IAM. »

Exemples

Pour consulter des exemples de politiques AWS IoT Events basées sur l'identité, consultez. AWS IoT Events exemples de politiques basées sur l'identité

AWS IoT Events politiques basées sur les ressources

AWS IoT Events ne prend pas en charge les politiques basées sur les ressources. » Pour afficher un exemple de page de stratégie basée sur les ressources détaillée, consultez https://docs.aws.amazon.com/lambda/latest/dg/access-control-resource-based.html.

Autorisation basée sur les AWS IoT Events tags

Vous pouvez associer des balises aux AWS IoT Events ressources ou transmettre des balises dans une demande à AWS IoT Events. Pour contrôler l'accès basé sur des étiquettes, vous devez fournir les informations d'étiquette dans l'élément de condition d'une politique utilisant les clés de condition `iotevents:ResourceTag/key-name`, `aws:RequestTag/key-name` ou `aws:TagKeys`. Pour plus d'informations sur le balisage des ressources AWS IoT Events , consultez Marquer vos ressources AWS IoT Events.

Pour afficher un exemple de stratégie basée sur l'identité permettant de limiter l'accès à une ressource basée sur les balises de cette ressource, veuillez consulter Afficher les AWS IoT Events entrées en fonction des balises.

AWS IoT Events Rôles IAM

Un rôle IAM est une entité au sein de vous Compte AWS qui possède des autorisations spécifiques.

Utilisation d'informations d'identification temporaires avec AWS IoT Events

Vous pouvez utiliser des informations d'identification temporaires pour vous connecter à l'aide de la fédération, endosser un rôle IAM ou encore pour endosser un rôle intercompte. Vous obtenez des informations d'identification de sécurité temporaires en appelant AWS Security Token Service (AWS STS) des opérations d'API telles que AssumeRoleou GetFederationToken.

AWS IoT Events ne prend pas en charge l'utilisation d'informations d'identification temporaires.

Rôles liés à un service

Les rôles liés aux AWS services permettent aux services d'accéder aux ressources d'autres services pour effectuer une action en votre nom. Les rôles liés à un service s'affichent dans votre compte IAM et sont la propriété du service. Un administrateur IAM peut consulter, mais ne peut pas modifier, les autorisations concernant les rôles liés à un service.

AWS IoT Events ne prend pas en charge les rôles liés à un service.

Rôles du service

Cette fonction permet à un service d'endosser une fonction du service en votre nom. Ce rôle autorise le service à accéder à des ressources d'autres services pour effectuer une action en votre nom. Les rôles de service s'affichent dans votre compte IAM et sont la propriété du compte. Cela signifie qu'un administrateur IAM peut modifier les autorisations associées à ce rôle. Toutefois, une telle action peut perturber le bon fonctionnement du service.

AWS IoT Events soutient les rôles de service.

AWS IoT Events exemples de politiques basées sur l'identité

Par défaut, les utilisateurs et les rôles ne sont pas autorisés à créer ou à modifier AWS IoT Events des ressources. Ils ne peuvent pas non plus effectuer de tâches à l'aide de l' AWS API AWS Management Console AWS CLI, ou. Un administrateur IAM doit créer des politiques IAM autorisant les utilisateurs et les rôles à exécuter des opérations d'API spécifiques sur les ressources spécifiées dont ils ont besoin. Il doit ensuite attacher ces stratégies aux utilisateurs ou aux groupes ayant besoin de ces autorisations.

Pour apprendre à créer une politique basée sur l'identité IAM à l'aide de ces exemples de documents de politique JSON, veuillez consulter Création de politiques dans l'onglet JSON dans le Guide de l'utilisateur IAM.

Rubriques

- Bonnes pratiques en matière de politiques
- Utilisation de la AWS IoT Events console
- Permettre aux utilisateurs de consulter leurs propres autorisations dans AWS IoT Events
- Accédez à une AWS IoT Events entrée
- Afficher les AWS IoT Events entrées en fonction des balises

Bonnes pratiques en matière de politiques

Les politiques basées sur l'identité sont très puissantes. Ils déterminent si quelqu'un peut créer, accéder ou supprimer AWS IoT Events des ressources dans votre compte. Ces actions peuvent entraîner des frais pour votre Compte AWS. Lorsque vous créez ou modifiez des politiques basées sur l'identité, suivez ces instructions et recommandations :

- Commencez à utiliser les politiques AWS gérées — Pour commencer à les utiliser AWS IoT Events rapidement, utilisez des politiques AWS gérées pour donner à vos employés les autorisations dont ils ont besoin. Ces politiques sont déjà disponibles dans votre compte et sont gérées et mises à jour par AWS. Pour plus d'informations, voir Commencer à utiliser les autorisations avec les politiques AWS gérées dans le Guide de l'utilisateur IAM.
- Accorder le privilège le plus faible : Lorsque vous créez des politiques personnalisées, accordez uniquement les autorisations requises pour exécuter une seule tâche. Commencez avec un ensemble d'autorisations minimum et accordez-en d'autres si nécessaire. Cette méthode est plus sûre que de commencer avec des autorisations trop permissives et d'essayer de les restreindre plus tard. Pour plus d'informations, consultez Accorder le moindre privilège possible dans le Guide de l'utilisateur IAM.
- Activer l'authentification multifactorielle pour les opérations sensibles — Pour plus de sécurité, demandez aux utilisateurs d'utiliser l'authentification multifactorielle (MFA) pour accéder aux ressources sensibles ou aux opérations d'API. Pour plus d'informations, consultez Utilisation de l'authentification multifacteur (MFA) dans AWS dans le Guide de l'utilisateur IAM.
- Utiliser des conditions de politique pour une plus grande sécurité : tant que cela reste pratique pour vous, définissez les conditions dans lesquelles vos politiques basées sur l'identité autorisent l'accès à une ressource. Par exemple, vous pouvez rédiger les conditions pour spécifier une plage d'adresses IP autorisées d'où peut provenir une demande. Vous pouvez également écrire des conditions pour autoriser les requêtes uniquement à une date ou dans une plage de temps spécifiée, ou pour imposer l'utilisation de SSL ou de MFA. Pour de plus amples informations, consultez Conditions pour éléments de politique JSON IAM dans le Guide de l'utilisateur IAM.

Utilisation de la AWS IoT Events console

Pour accéder à la AWS IoT Events console, vous devez disposer d'un ensemble minimal d'autorisations. Ces autorisations doivent vous permettre de répertorier et d'afficher les détails AWS IoT Events des ressources de votre Compte AWS. Si vous créez une politique basée sur l'identité qui est plus restrictive que l'ensemble minimum d'autorisations requis, la console ne fonctionnera pas comme prévu pour les entités (utilisateurs ou rôles) tributaires de cette politique.

Pour garantir que ces entités peuvent toujours utiliser la AWS IoT Events console, associez également la politique AWS gérée suivante aux entités. Pour plus d'informations, consultez la section Ajouter des autorisations à un utilisateur dans le guide de l'utilisateur IAM :

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"
        }
    ]
}
```

Il n'est pas nécessaire d'accorder des autorisations de console minimales aux utilisateurs qui appellent uniquement l'API AWS CLI ou l' AWS API. Autorisez plutôt l’accès à uniquement aux actions qui correspondent à l’opération d’API que vous tentez d’effectuer.

Permettre aux utilisateurs de consulter leurs propres autorisations dans AWS IoT Events

Cet exemple montre comment créer une stratégie qui permet aux utilisateurs d'afficher les stratégies en ligne et gérées attachées à leur identité d'utilisateur. Permettre aux utilisateurs de consulter leurs propres autorisations IAM est utile pour les sensibiliser à la sécurité et les fonctionnalités de libre-service. Cette politique inclut les autorisations permettant d'effectuer cette action sur la console ou par programmation à l'aide de l'API AWS CLI or AWS .

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": "*"
```

```
            }
        ]
    }
```

Accédez à une AWS IoT Events entrée

Le contrôle d'accès granulaire aux AWS IoT Events entrées est important pour maintenir la sécurité dans les environnements multi-utilisateurs ou multi-équipes. Cette section explique comment créer des politiques IAM qui accordent l'accès à des AWS IoT Events entrées spécifiques tout en restreignant l'accès à d'autres.

Dans cet exemple, vous pouvez autoriser un utilisateur à Compte AWS accéder à l'une de vos AWS IoT Events entrées,`exampleInput`. Vous pouvez également autoriser l'utilisateur à ajouter, mettre à jour et supprimer des entrées.

La stratégie accorde à l'utilisateur les autorisations `iotevents:ListInputs`, `iotevents:DescribeInput`, `iotevents:CreateInput`, `iotevents:DeleteInput` et `iotevents:UpdateInput`. Pour un exemple de présentation de l'Amazon Simple Storage Service (Amazon S3) qui accorde des autorisations aux utilisateurs et les teste à l'aide de la console, consultez la section Contrôle de l'accès à un compartiment avec des politiques utilisateur.

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/*"
        }
    ]
}
```

Afficher les AWS IoT Events entrées en fonction des balises

Les balises vous aident à organiser AWS IoT Events les ressources. Vous pouvez utiliser des conditions dans votre politique basée sur l'identité pour contrôler l'accès aux AWS IoT Events ressources en fonction de balises. Cet exemple montre comment créer une politique permettant d'afficher un*input*. Toutefois, l'autorisation est accordée uniquement si la balise `Owner` *input* a la valeur du nom d'utilisateur de cet utilisateur. Cette politique accorde également les autorisations nécessaires pour réaliser cette action sur la console.

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}"}
            }
        }
    ]
}
```

Vous pouvez attacher cette stratégie aux utilisateurs de votre compte. Si un utilisateur nommé `richard-roe` tente de consulter un AWS IoT Events *`input`*, celui-ci *`input`* doit être marqué `Owner=richard-roe` ou`owner=richard-roe`. Dans le cas contraire, l'utilisateur se voit refuser l'accès. La clé de condition d'étiquette `Owner` correspond à la fois à `Owner` et à `owner`, car les noms de clé de condition ne sont pas sensibles à la casse. Pour plus d'informations, consultez Conditions pour éléments de politique JSON IAM dans le Guide de l'utilisateur IAM.

Prévention interservices confuse des adjoints pour AWS IoT Events

> **Note**
>
> - Le AWS IoT Events service vous permet uniquement d'utiliser des rôles pour démarrer des actions dans le même compte dans lequel une ressource a été créée. Cela permet d'éviter une attaque adjointe confuse AWS IoT Events.
> - Cette page vous sert de référence pour voir comment fonctionne le problème de confusion des adjoints et peut être évité si les ressources entre comptes étaient autorisées dans le AWS IoT Events service.

Le problème de député confus est un problème de sécurité dans lequel une entité qui n'est pas autorisée à effectuer une action peut contraindre une entité plus privilégiée à le faire. En AWS, l'usurpation d'identité interservices peut entraîner la confusion des adjoints.

L'usurpation d'identité entre services peut se produire lorsqu'un service (le service appelant) appelle un autre service (le service appelé). Le service appelant peut être manipulé et ses autorisations utilisées pour agir sur les ressources d'un autre client auxquelles on ne serait pas autorisé à accéder autrement. Pour éviter cela, AWS fournit des outils qui vous aident à protéger vos données pour tous les services auprès des principaux fournisseurs de services qui ont obtenu l'accès aux ressources de votre compte.

Nous recommandons d'utiliser les clés de contexte de condition `aws:SourceAccount`globale `aws:SourceArn`et les clés contextuelles dans les politiques de ressources afin de limiter les autorisations qui AWS IoT Events accordent un autre service à la ressource. Si la valeur `aws:SourceArn` ne contient pas l'ID du compte, tel qu'un ARN de compartiment Amazon S3, vous devez utiliser les deux clés de contexte de condition globale pour limiter les autorisations. Si vous utilisez les deux clés de contexte de condition globale et que la valeur `aws:SourceArn` contient l'ID de compte, la valeur `aws:SourceAccount` et le compte dans la valeur `aws:SourceArn` doivent utiliser le même ID de compte lorsqu'ils sont utilisés dans la même instruction de politique.

Utilisez `aws:SourceArn` si vous souhaitez qu'une seule ressource soit associée à l'accès entre services. Utilisez `aws:SourceAccount` si vous souhaitez autoriser l'association d'une ressource de ce compte à l'utilisation interservices. La valeur de `aws:SourceArn` doit être le modèle de détecteur ou le modèle d'alarme associé à la `sts:AssumeRole` demande.

Le moyen le plus efficace de se protéger contre le problème de député confus consiste à utiliser la clé de contexte de condition globale `aws:SourceArn` avec l'ARN complet de la ressource. Si vous ne connaissez pas l'ARN complet de la ressource ou si vous spécifiez plusieurs ressources, utilisez la clé de contexte de condition globale `aws:SourceArn` avec des caractères génériques (*) pour les parties inconnues de l'ARN. Par exemple, `arn:aws:`*`iotevents`*`:*:`*`123456789012`*`:*`.

Les exemples suivants montrent comment utiliser les touches de contexte de condition `aws:SourceAccount` globale `aws:SourceArn` et globale AWS IoT Events pour éviter le problème de confusion des adjoints.

Rubriques

- Exemple : accès sécurisé à un modèle AWS IoT Events de détecteur
- Exemple : accès sécurisé à un modèle AWS IoT Events d'alarme
- Exemple : accéder à une AWS IoT Events ressource dans une région spécifiée
- Exemple : configurer les options de journalisation pour AWS IoT Events

Exemple : accès sécurisé à un modèle AWS IoT Events de détecteur

Cet exemple montre comment créer une politique IAM qui accorde en toute sécurité l'accès à un modèle de détecteur spécifique dans AWS IoT Events. La politique utilise des conditions pour garantir que seuls le AWS compte et le AWS IoT Events service spécifiés peuvent assumer le rôle, ajoutant ainsi un niveau de sécurité supplémentaire. Dans cet exemple, le rôle peut uniquement accéder au modèle de détecteur nommé*`WindTurbine01`*.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": [
                    "iotevents.amazonaws.com"
                ]
            },
            "Action": "sts:AssumeRole",
            "Condition": {
                "StringEquals": {
                "aws:SourceAccount": "123456789012"
                },
                "ArnEquals": {
                "aws:SourceArn": "arn:aws:iotevents:us-
east-1:123456789012:detectorModel/WindTurbine01"
                }
            }
        }
    ]
}
```

Exemple : accès sécurisé à un modèle AWS IoT Events d'alarme

Cet exemple montre comment créer une politique IAM qui permet d'accéder en toute sécurité AWS IoT Events aux modèles d'alarme. La politique utilise des conditions pour garantir que seuls le AWS compte et le AWS IoT Events service spécifiés peuvent assumer le rôle.

Dans cet exemple, le rôle peut accéder à n'importe quel modèle d'alarme au sein du AWS compte spécifié, comme indiqué par le * caractère générique dans l'ARN du modèle d'alarme. Les `aws:SourceArn` conditions `aws:SourceAccount` et les conditions fonctionnent ensemble pour éviter le problème confus des adjoints.

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/*"
                }
            }
        }
    ]
}
```

Exemple : accéder à une AWS IoT Events ressource dans une région spécifiée

Cet exemple montre comment configurer un rôle IAM pour accéder aux AWS IoT Events ressources d'une AWS région spécifique. En utilisant des politiques IAM spécifiques ARNs à une région, vous pouvez restreindre l'accès aux AWS IoT Events ressources dans différentes zones géographiques. Cette approche peut aider à maintenir la sécurité et la conformité dans les déploiements multirégionaux. Dans cet exemple, la région est*us-east-1*.

JSON

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

```
                ]
            },
            "Action": "sts:AssumeRole",
            "Condition": {
                "StringEquals": {
                "aws:SourceAccount": "123456789012"
                },
                "ArnEquals": {
                "aws:SourceArn": "arn:aws:iotevents:us-east-1:123456789012:*"
                }
            }
        }
    ]
}
```

Exemple : configurer les options de journalisation pour AWS IoT Events

Une journalisation appropriée est importante pour la surveillance, le débogage et l'audit de vos AWS IoT Events applications. Cette section fournit un aperçu des options de journalisation disponibles dans AWS IoT Events.

Cet exemple montre comment configurer un rôle IAM qui permet de AWS IoT Events consigner des données dans CloudWatch Logs. L'utilisation de caractères génériques (*) dans l'ARN de la ressource permet une journalisation complète de votre AWS IoT Events infrastructure.

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:*"
                }
            }
        }
    ]
}
```

Résoudre les problèmes d' AWS IoT Events identité et d'accès

Utilisez les informations suivantes pour vous aider à diagnostiquer et à résoudre les problèmes courants que vous pouvez rencontrer lorsque vous travaillez avec AWS IoT Events IAM.

Rubriques

- Je ne suis pas autorisé à effectuer une action dans AWS IoT Events
- Je ne suis pas autorisé à effectuer iam:PassRole
- Je souhaite permettre à des personnes extérieures Compte AWS à moi d'accéder à mes AWS IoT Events ressources

Je ne suis pas autorisé à effectuer une action dans AWS IoT Events

S'il vous AWS Management Console indique que vous n'êtes pas autorisé à effectuer une action, vous devez contacter votre administrateur pour obtenir de l'aide. Votre administrateur est la personne qui vous a fourni votre nom d'utilisateur et votre mot de passe.

L'exemple d'erreur suivant se produit lorsque l'utilisateur IAM `mateojackson` tente d'utiliser la console pour afficher des informations détaillées concernant un élément *`input`* mais ne dispose pas des autorisations `iotevents:`*`ListInputs`* nécessaires.

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

Dans ce cas, Mateo demande à son administrateur de mettre à jour ses politiques pour lui permettre d'accéder à la ressource *`my-example-input`* à l'aide de l'action `iotevents:`*`ListInput`*.

Je ne suis pas autorisé à effectuer `iam:PassRole`

Si vous recevez une erreur selon laquelle vous n'êtes pas autorisé à exécuter `iam:PassRole` l'action, vos stratégies doivent être mises à jour afin de vous permettre de transmettre un rôle à AWS IoT Events.

Certains vous Services AWS permettent de transmettre un rôle existant à ce service au lieu de créer un nouveau rôle de service ou un rôle lié à un service. Pour ce faire, vous devez disposer des autorisations nécessaires pour transmettre le rôle au service.

L'exemple d'erreur suivant se produit lorsqu'un utilisateur IAM nommé `marymajor` essaie d'utiliser la console pour exécuter une action dans AWS IoT Events. Toutefois, l'action nécessite que le service ait des autorisations accordées par une fonction de service. Mary n'est pas autorisée à transmettre le rôle au service.

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

Dans ce cas, les politiques de Mary doivent être mises à jour pour lui permettre d'exécuter l'action `iam:PassRole`.

Si vous avez besoin d'aide, contactez votre AWS administrateur. Votre administrateur vous a fourni vos informations d'identification de connexion.

Je souhaite permettre à des personnes extérieures Compte AWS à moi d'accéder à mes AWS IoT Events ressources

Vous pouvez créer un rôle que les utilisateurs provenant d'autres comptes ou les personnes extérieures à votre organisation pourront utiliser pour accéder à vos ressources. Vous pouvez spécifier qui est autorisé à assumer le rôle. Pour les services qui prennent en charge les politiques basées sur les ressources ou les listes de contrôle d'accès (ACLs), vous pouvez utiliser ces politiques pour autoriser les utilisateurs à accéder à vos ressources.

Consultez les rubriques suivantes pour déterminer les meilleures options :

- Pour savoir si ces fonctionnalités sont prises AWS IoT Events en charge, consultezComment AWS IoT Events fonctionne avec IAM.
- Pour savoir comment fournir l'accès à vos ressources sur celles Comptes AWS que vous possédez, consultez la section Fournir l'accès à un utilisateur IAM dans un autre utilisateur Compte AWS que vous possédez dans le Guide de l'utilisateur IAM.

- Pour savoir comment fournir l'accès à vos ressources à des tiers Comptes AWS, consultez la section Fournir un accès à des ressources Comptes AWS détenues par des tiers dans le guide de l'utilisateur IAM.
- Pour savoir comment fournir un accès par le biais de la fédération d'identité, consultez Fournir un accès à des utilisateurs authentifiés en externe (fédération d'identité) dans le Guide de l'utilisateur IAM.
- Pour en savoir plus sur la différence entre l'utilisation des rôles et des politiques basées sur les ressources pour l'accès intercompte, consultez Accès intercompte aux ressources dans IAM dans le Guide de l'utilisateur IAM.

Surveillance AWS IoT Events pour maintenir la fiabilité, la disponibilité et les performances

La surveillance joue un rôle important dans le maintien de la fiabilité, de la disponibilité AWS IoT Events et des performances de vos AWS solutions. Vous devez collecter des données de surveillance provenant de toutes les parties de votre AWS solution afin de pouvoir corriger plus facilement une défaillance multipoint, le cas échéant. Avant de commencer la surveillance AWS IoT Events, vous devez créer un plan de surveillance qui inclut les réponses aux questions suivantes :

- Quels sont les objectifs de la surveillance ?
- Quelles sont les ressources à surveiller ?
- A quelle fréquence les ressources doivent-elles être surveillées ?
- Quels outils de surveillance utiliser ?
- Qui exécute les tâches de supervision ?
- Qui doit être informé en cas de problème ?

L'étape suivante consiste à établir une base de référence pour des AWS IoT Events performances normales dans votre environnement, en mesurant les performances à différents moments et dans différentes conditions de charge. Lorsque vous surveillez AWS IoT Events, conservez les données d'historique de surveillance afin de pouvoir les comparer aux données de performances actuelles, d'identifier les modèles de performances normales et les anomalies de performances, et de concevoir des méthodes pour résoudre les problèmes.

Par exemple, si vous utilisez Amazon EC2, vous pouvez surveiller l'utilisation du processeur, du disque I/O, and network utilization for your instances. When performance falls outside your

established baseline, you might need to reconfigure or optimize the instance to reduce CPU utilization, improve disk I/O ou réduire le trafic réseau.

Rubriques

- Outils disponibles pour surveiller AWS IoT Events
- Surveillance AWS IoT Events avec Amazon CloudWatch
- Journalisation des appels d' AWS IoT Events API avec AWS CloudTrail

Outils disponibles pour surveiller AWS IoT Events

AWS fournit divers outils que vous pouvez utiliser pour surveiller AWS IoT Events. Vous pouvez configurer certains outils pour qu'ils effectuent la supervision automatiquement, tandis que d'autres nécessitent une intervention manuelle. Nous vous recommandons d'automatiser le plus possible les tâches de supervision.

Outils de surveillance automatique

Vous pouvez utiliser les outils de surveillance automatique suivants pour surveiller AWS IoT Events et signaler tout problème :

- Amazon CloudWatch Logs — Surveillez, stockez et accédez à vos fichiers journaux depuis AWS CloudTrail ou d'autres sources. Pour plus d'informations, consultez la section Utilisation CloudWatch des tableaux de bord Amazon dans le guide de CloudWatch l'utilisateur Amazon.
- AWS CloudTrail Surveillance des journaux : partagez des fichiers journaux entre comptes, surveillez les fichiers CloudTrail journaux en temps réel en les envoyant à CloudWatch Logs, écrivez des applications de traitement des journaux en Java et vérifiez que vos fichiers journaux n'ont pas changé après leur livraison par. CloudTrail Pour plus d'informations, consultez la section Utilisation des fichiers CloudTrail journaux dans le Guide de AWS CloudTrail l'utilisateur.

Outils de surveillance manuelle

Une autre partie importante de la surveillance AWS IoT Events consiste à surveiller manuellement les éléments non couverts par les CloudWatch alarmes. Le AWS IoT Events tableau de bord de AWS console et les autres tableaux de bord fournissent une at-a-glance vue de l'état de votre AWS environnement. CloudWatch Nous vous recommandons de vérifier également les fichiers journaux AWS IoT Events.

- La AWS IoT Events console affiche :
  - Modèles de détecteur
  - Détecteurs
  - Inputs
  - Settings
- La page d' CloudWatch accueil indique :
  - Alarmes et statuts en cours
  - Graphiques des alarmes et des ressources
  - Statut d'intégrité du service

  En outre, vous pouvez utiliser CloudWatch pour effectuer les opérations suivantes :
  - Création Création d'un CloudWatch tableau de bord pour surveiller les services qui vous intéressent
  - Représenter graphiquement les données de métriques pour résoudre les problèmes et découvrir les tendances
  - Recherchez et parcourez tous les indicateurs de vos AWS ressources
  - Créer et modifier des alarmes pour être informé des problèmes

Surveillance AWS IoT Events avec Amazon CloudWatch

Lorsque vous développez ou débugguez un modèle de AWS IoT Events détecteur, vous devez savoir ce qu'il AWS IoT Events fait et quelles sont les erreurs qu'il rencontre. Amazon CloudWatch surveille vos AWS ressources et les applications que vous utilisez AWS en temps réel. Vous bénéficiez CloudWatch ainsi d'une visibilité à l'échelle du système sur l'utilisation des ressources, les performances des applications et la santé opérationnelle. Activer la CloudWatch journalisation Amazon lors du développement AWS IoT Events de modèles de détecteurscontient des informations sur la façon d'activer la CloudWatch journalisation pour AWS IoT Events. Pour générer des journaux tels que celui illustré ci-dessous, vous devez définir le niveau de verbosité sur « Déboguer » et fournir une ou plusieurs cibles de débogage sous forme de nom de modèle de détecteur et en option. KeyValue

L'exemple suivant montre une entrée de journal de niveau CloudWatch DEBUG générée par AWS IoT Events.

```
{
```

```
  "timestamp": "2019-03-15T15:56:29.412Z",
  "level": "DEBUG",
  "logMessage": "Summary of message evaluation",
  "context": "MessageEvaluation",
  "status": "Success",
  "messageId": "SensorAggregate_2th846h",
  "keyValue": "boiler_1",
  "detectorModelName": "BoilerAlarmDetector",
  "initialState": "high_temp_alarm",
  "initialVariables": {
    "high_temp_count": 1,
    "high_pressure_count": 1
  },
  "finalState": "no_alarm",
  "finalVariables": {
    "high_temp_count": 0,
    "high_pressure_count": 0
  },
  "message": "{ \"temp\": 34.9, \"pressure\": 84.5}",
  "messageType": "CUSTOMER_MESSAGE",
  "conditionEvaluationResults": [
    {
      "result": "True",
      "eventName": "alarm_cleared",
      "state": "high_temp_alarm",
      "lifeCycle": "OnInput",
      "hasTransition": true
    },
    {
      "result": "Skipped",
      "eventName": "alarm_escalated",
      "state": "high_temp_alarm",
      "lifeCycle": "OnInput",
      "hasTransition": true,
      "resultDetails": "Skipped due to transition from alarm_cleared event"
    },
    {
      "result": "True",
      "eventName": "should_recall_technician",
      "state": "no_alarm",
      "lifeCycle": "OnEnter",
      "hasTransition": true
    }
  ]
```

```
}
```

Journalisation des appels d' AWS IoT Events API avec AWS CloudTrail

AWS IoT Events est intégré à AWS CloudTrail un service qui fournit un enregistrement des actions entreprises par un utilisateur, un rôle ou un AWS service dans AWS IoT Events. CloudTrail capture tous les appels d'API AWS IoT Events sous forme d'événements, y compris les appels depuis la AWS IoT Events console et les appels de code vers le AWS IoT Events APIs.

Si vous créez un suivi, vous pouvez activer la diffusion continue d' CloudTrail événements vers un compartiment Amazon S3, y compris les événements pour AWS IoT Events. Si vous ne configurez pas de suivi, vous pouvez toujours consulter les événements les plus récents dans la CloudTrail console dans Historique des événements. À l'aide des informations collectées par CloudTrail, vous pouvez déterminer la demande qui a été faite AWS IoT Events, l'adresse IP à partir de laquelle la demande a été faite, qui a fait la demande, quand elle a été faite et des détails supplémentaires.

Pour en savoir plus CloudTrail, consultez le guide de AWS CloudTrail l'utilisateur.

AWS IoT Events informations dans CloudTrail

CloudTrail est activé sur votre AWS compte lorsque vous le créez. Lorsqu'une activité se produit dans AWS IoT Events, cette activité est enregistrée dans un CloudTrail événement avec d'autres événements AWS de service dans l'historique des événements. Vous pouvez consulter, rechercher et télécharger les événements récents dans votre AWS compte. Pour plus d'informations, consultez la section Utilisation de l'historique des CloudTrail événements.

Pour un enregistrement continu des événements de votre AWS compte, y compris des événements pour AWS IoT Events, créez un parcours. Un suivi permet CloudTrail de fournir des fichiers journaux à un compartiment Amazon S3. Par défaut, lorsque vous créez un parcours dans la console, celui-ci s'applique à toutes les AWS régions. Le journal enregistre les événements de toutes les régions de la AWS partition et transmet les fichiers journaux au compartiment Amazon S3 que vous spécifiez. En outre, vous pouvez configurer d'autres AWS services pour analyser et agir de manière plus approfondie sur les données d'événements collectées dans CloudTrail les journaux. Pour en savoir plus, consultez :

- Création d'un parcours pour votre AWS compte
- Intégrations et services pris en charge par CloudTrail
- Configuration des notifications Amazon SNS pour CloudTrail

- Réception de fichiers CloudTrail journaux de plusieurs régions et réception de fichiers CloudTrail journaux de plusieurs comptes

Chaque événement ou entrée de journal contient des informations sur la personne ayant initié la demande. Les informations relatives à l'identité permettent de déterminer les éléments suivants :

- Si la demande a été effectuée avec des informations d'identification d'utilisateur root ou IAM.
- Si la demande a été effectuée avec des informations d'identification de sécurité temporaires pour un rôle ou un utilisateur fédéré.
- Si la demande a été faite par un autre AWS service.

Pour plus d'informations, consultez l'élément CloudTrailUserIdentity. AWS IoT Events les actions sont documentées dans la référence de AWS IoT Events l'API.

Comprendre les entrées du fichier AWS IoT Events journal

Un suivi est une configuration qui permet de transmettre des événements sous forme de fichiers journaux à un compartiment Amazon S3 que vous spécifiez. AWS CloudTrail les fichiers journaux contiennent une ou plusieurs entrées de journal. Un événement représente une demande unique provenant de n'importe quelle source et inclut des informations sur l'action demandée, la date et l'heure de l'action, les paramètres de la demande, etc. CloudTrail les fichiers journaux ne constituent pas une trace ordonnée des appels d'API publics, ils n'apparaissent donc pas dans un ordre spécifique.

Lorsque la CloudTrail journalisation est activée dans votre AWS compte, la plupart des appels d'API effectués vers AWS IoT Events des actions sont suivis dans des fichiers CloudTrail journaux où ils sont écrits avec d'autres enregistrements de AWS service. CloudTraildétermine à quel moment créer et écrire dans un nouveau fichier en fonction d'une période et de la taille du fichier.

Chaque entrée du journal contient des informations sur la personne qui a généré la demande. Les informations d'identité de l'utilisateur dans l'entrée de journal permettent de déterminer les éléments suivants :

- Si la demande a été effectuée avec des informations d'identification d'utilisateur root ou IAM.
- Si la demande a été effectuée avec des informations d'identification de sécurité temporaires pour un rôle ou un utilisateur fédéré.
- Si la demande a été faite par un autre AWS service.

Vous pouvez stocker vos fichiers journaux dans votre compartiment Amazon S3 aussi longtemps que vous le souhaitez, mais vous pouvez également définir des règles de cycle de vie Amazon S3 pour archiver ou supprimer automatiquement les fichiers journaux. Par défaut, vos fichiers journaux sont chiffrés avec le chiffrement côté serveur (SSE) d'Amazon S3.

Pour être averti lors de la livraison des fichiers journaux, vous pouvez configurer CloudTrail pour publier des notifications Amazon SNS lorsque de nouveaux fichiers journaux sont livrés. Pour plus d'informations, consultez Configuration des notifications Amazon SNS pour CloudTrail.

Vous pouvez également agréger les fichiers AWS IoT Events journaux de plusieurs AWS régions et de plusieurs AWS comptes dans un seul compartiment Amazon S3.

Pour plus d'informations, consultez les sections Réception de fichiers CloudTrail journaux de plusieurs régions et Réception de fichiers CloudTrail journaux de plusieurs comptes.

Exemple : DescribeDetector action pour CloudTrail

L'exemple suivant montre une entrée de CloudTrail journal illustrant l'`DescribeDetector`action.


```
{
  "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"
}
```

Exemple : CreateDetectorModel action pour CloudTrail

L'exemple suivant montre une entrée de CloudTrail journal illustrant l'`CreateDetectorModel`action.

```
{
  "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"
}
```


Exemple : CreateInput action pour CloudTrail

L'exemple suivant montre une entrée de CloudTrail journal illustrant l'`CreateInput`action.


```
{
  "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"
}
```

Exemple : DeleteDetectorModel action pour CloudTrail

L'exemple suivant montre une entrée de CloudTrail journal illustrant l'`DeleteDetectorModel`action.

```
{
  "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"
}
```

Exemple : DeleteInput action pour CloudTrail

L'exemple suivant montre une entrée de CloudTrail journal illustrant l'`DeleteInput`action.

```
{
  "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"
}
```

Exemple : DescribeDetectorModel action pour CloudTrail

L'exemple suivant montre une entrée de CloudTrail journal illustrant l'`DescribeDetectorModel`action.

```
{
  "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"
}
```

Exemple : DescribeInput action pour CloudTrail

L'exemple suivant montre une entrée de CloudTrail journal illustrant l'`DescribeInput`action.

```
{
  "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"
}
```

Exemple : DescribeLoggingOptions action pour CloudTrail

L'exemple suivant montre une entrée de CloudTrail journal illustrant l'`DescribeLoggingOptions`action.

```
{
  "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"
}
```

Exemple : ListDetectorModels action pour CloudTrail

L'exemple suivant montre une entrée de CloudTrail journal illustrant l'`ListDetectorModels`action.

```
{
  "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"
}
```

Exemple : ListDetectorModelVersions action pour CloudTrail

L'exemple suivant montre une entrée de CloudTrail journal illustrant l'`ListDetectorModelVersions`action.

```
{
  "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"
}
```

Exemple : ListDetectors action pour CloudTrail

L'exemple suivant montre une entrée de CloudTrail journal illustrant l'`ListDetectors`action.

```
{
  "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"
}
```

Exemple : ListInputs action pour CloudTrail

L'exemple suivant montre une entrée de CloudTrail journal illustrant l'`ListInputs`action.

```
{
  "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"
}
```

Exemple : PutLoggingOptions action pour CloudTrail

L'exemple suivant montre une entrée de CloudTrail journal illustrant l'`PutLoggingOptions`action.

```
{
  "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"
}
```

Exemple : UpdateDetectorModel action pour CloudTrail

L'exemple suivant montre une entrée de CloudTrail journal illustrant l'`UpdateDetectorModel`action.

```
{
  "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"
}
```

Exemple : UpdateInput action pour CloudTrail

L'exemple suivant montre une entrée de CloudTrail journal illustrant l'`UpdateInput`action.

```
{
  "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"
}
```

Exemple : BatchPutMessage action pour CloudTrail

AWS IoT Events peut utiliser une CloudTrail intégration pour la journalisation de l'API du plan de données. Cet exemple ajoute des détails sur les événements liés aux données par le biais de l'`BatchPutMessage`action.


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


Validation de conformité pour AWS IoT Events

Pour savoir si un programme Services AWS de conformité Service AWS s'inscrit dans le champ d'application de programmes de conformité spécifiques, consultez Services AWS la section de conformité et sélectionnez le programme de conformité qui vous intéresse. Pour des informations générales, voir Programmes de AWS conformité Programmes AWS de .

Vous pouvez télécharger des rapports d'audit tiers à l'aide de AWS Artifact. Pour plus d'informations, voir Téléchargement de rapports dans AWS Artifact .

Votre responsabilité en matière de conformité lors de l'utilisation Services AWS est déterminée par la sensibilité de vos données, les objectifs de conformité de votre entreprise et les lois et réglementations applicables. Pour plus d'informations sur votre responsabilité en matière de conformité lors de l'utilisation Services AWS, consultez AWS la documentation de sécurité.

Résilience dans AWS IoT Events

L'infrastructure AWS mondiale est construite autour des AWS régions et des zones de disponibilité. AWS Les régions fournissent plusieurs zones de disponibilité physiquement séparées et isolées, reliées par un réseau à latence faible, à haut débit et hautement redondant. Avec les zones de disponibilité, vous pouvez concevoir et exploiter des applications et des bases de données qui basculent automatiquement d'une zone de disponibilité à l'autre sans interruption. Les zones de disponibilité sont plus hautement disponibles, tolérantes aux pannes et évolutives que les infrastructures traditionnelles à un ou plusieurs centres de données.

Pour plus d'informations sur AWS les régions et les zones de disponibilité, consultez la section Infrastructure AWS globale.

Sécurité de l'infrastructure dans AWS IoT Events

En tant que service géré, AWS IoT Events il est protégé par la sécurité du réseau AWS mondial. Pour plus d'informations sur les services AWS de sécurité et sur la manière dont AWS l'infrastructure est protégée, consultez la section Sécurité du AWS cloud. Pour concevoir votre AWS environnement en utilisant les meilleures pratiques en matière de sécurité de l'infrastructure, consultez la section Protection de l'infrastructure dans le cadre AWS bien architecturé du pilier de sécurité.

Vous utilisez des appels d'API AWS publiés pour accéder AWS IoT Events via le réseau. Les clients doivent prendre en charge les éléments suivants :

- Protocole TLS (Transport Layer Security). Nous exigeons TLS 1.2 et recommandons TLS 1.3.
- Ses suites de chiffrement PFS (Perfect Forward Secrecy) comme DHE (Ephemeral Diffie-Hellman) ou ECDHE (Elliptic Curve Ephemeral Diffie-Hellman). La plupart des systèmes modernes tels que Java 7 et les versions ultérieures prennent en charge ces modes.

AWS quotas de service pour les AWS IoT Events ressources

Le Références générales AWS Guide fournit les quotas par défaut AWS IoT Events pour un AWS compte. Sauf indication contraire, chaque quota est établi par AWS région. Pour plus d'informations, consultez les sections AWS IoT Events Points de terminaison et quotas et Quotas de AWS Service dans le Références générales AWS Guide.

Pour demander une augmentation du quota de service, soumettez un dossier d'assistance dans la console du centre de support. Pour plus d'informations, consultez Demande d'augmentation de quota dans le Guide de l'utilisateur Service Quotas.

> **Note**
>
> - Tous les noms des modèles de détecteurs et des entrées doivent être uniques au sein d'un compte.
> - Vous ne pouvez pas modifier les noms des modèles de détecteurs et des entrées une fois qu'ils ont été créés.

Marquer vos ressources AWS IoT Events

Pour vous aider à gérer et à organiser vos modèles de détecteurs et vos entrées, vous pouvez éventuellement attribuer vos propres métadonnées à chacune de ces ressources sous forme de balises. Cette section décrit les balises et vous montre comment les créer.

Principes de base des étiquettes

Les balises vous permettent de classer vos AWS IoT Events ressources de différentes manières, par exemple par objectif, propriétaire ou environnement. Cela est utile lorsque vous avez de nombreuses ressources du même type. Vous pouvez identifier rapidement une ressource spécifique en fonction des étiquettes que vous lui avez attribuées.

Chaque balise est constituée d'une clé et d'une valeur facultative que vous définissez. Par exemple, vous pouvez définir un ensemble de balises pour vos entrées afin de suivre les appareils qui envoient ces entrées par type. Nous vous recommandons de créer un ensemble de clés de balise répondant à vos besoins pour chaque type de ressource. L'utilisation d'un ensemble de clés de balise cohérent facilite la gestion de vos ressources.

Vous pouvez rechercher et filtrer les ressources en fonction des balises que vous ajoutez ou appliquez, utiliser des balises pour classer et suivre vos coûts, et également utiliser des balises pour contrôler l'accès à vos ressources, comme décrit dans la section Utilisation des balises avec les politiques IAM dans le Guide du AWS IoT développeur.

Pour faciliter l'utilisation, l'éditeur de balises AWS Management Console fournit un moyen centralisé et unifié de créer et de gérer vos balises. Pour plus d'informations, consultez la section Mise en route avec l'éditeur de balises dans le guide de l'utilisateur AWS des ressources de balisage et de l'éditeur de balises.

Vous pouvez également travailler avec des balises à l'aide de l'API AWS CLI et de l' AWS IoT Events API. Vous pouvez associer des balises à des modèles de détecteurs et à des entrées lorsque vous les créez en utilisant le `"Tags"` champ des commandes suivantes :

- CreateDetectorModel
- CreateInput

Vous pouvez ajouter, modifier ou supprimer des balises pour les ressources existantes qui prennent en charge le balisage à l'aide des commandes suivantes :

- TagResource
- ListTagsForResource
- UntagResource

Vous pouvez modifier les clés et valeurs de balise, et vous pouvez retirer des balises d'une ressource à tout moment. Vous pouvez définir la valeur d'une balise sur une chaîne vide, mais vous ne pouvez pas définir la valeur d'une balise sur null. Si vous ajoutez une balise ayant la même clé qu'une balise existante sur cette ressource, la nouvelle valeur remplace l'ancienne valeur. Si vous supprimez une ressource, toutes les balises associées à celle-ci sont également supprimées.

Pour plus d'informations, voir Meilleures pratiques pour le balisage des ressources AWS

Limites et restrictions liées aux balises

Les restrictions de base suivantes s'appliquent aux balises :

- Nombre maximal de balises par ressource : 50
- Longueur de clé maximale : 127 caractères Unicode en UTF-8
- Longueur maximale de la valeur : 255 caractères Unicode en UTF-8
- Les clés et les valeurs des balises distinguent les majuscules et minuscules.
- N'utilisez pas le "`aws:`" préfixe dans les noms ou les valeurs de vos balises, car il est réservé à l' AWS usage. Vous ne pouvez pas modifier ou supprimer des noms ou valeurs de balise ayant ce préfixe. Les balises avec ce préfixe ne sont pas comptabilisées comme vos balises pour la limite de ressources.
- Si votre schéma de balisage est utilisé pour plusieurs services et ressources , n'oubliez pas que d'autres services peuvent avoir des restrictions concernant les caractères autorisés. En général, les caractères autorisés sont les lettres, les espaces et les chiffres représentables en UTF-8, ainsi que les caractères spéciaux suivants : + - = . _ : / @.

Utilisation des balises avec des politiques IAM

Vous pouvez appliquer des autorisations de niveau ressource basées sur des balises dans les stratégies IAM que vous utilisez pour les actions d'API AWS IoT Events . Vous bénéficiez ainsi d'un meilleur contrôle sur les ressources qu'un utilisateur peut créer, modifier ou utiliser.

Vous pouvez utiliser l'élément `Condition` (également appelé bloc `Condition`) avec les clés et valeurs de contexte de condition suivantes dans une politique IAM pour contrôler l'accès des utilisateurs (autorisations) en fonction des balises d'une ressource :

- Utilisez `aws:ResourceTag/<tag-key>: <tag-value>` pour accorder ou refuser aux utilisateurs des actions sur des ressources ayant des balises spécifiques.
- Utilisez `aws:RequestTag/<tag-key>: <tag-value>` pour exiger qu'une balise spécifique soit utilisée (ou ne soit pas utilisée) lorsque vous effectuez une demande d'API pour créer ou modifier une ressource qui autorise les balises.
- Utilisez `aws:TagKeys: [<tag-key>, ...]` pour exiger qu'un ensemble de clés de balise spécifique soit utilisé (ou ne soit pas utilisé) lorsque vous effectuez une demande d'API pour créer ou modifier une ressource qui autorise les balises.

> **Note**
>
> Les clés et les valeurs de contexte de condition dans une politique IAM s'appliquent uniquement aux actions AWS IoT Events dans lesquelles un identifiant pour une ressource pouvant être balisée est un paramètre obligatoire.

Le contrôle de l'accès à l'aide de balises dans le guide de Gestion des identités et des accès AWS l'utilisateur contient des informations supplémentaires sur l'utilisation des balises. La section Référence de politique JSON IAM de ce guide fournit la syntaxe détaillée, des descriptions, ainsi que des exemples des éléments, des variables et de la logique d'évaluation des politiques JSON dans IAM.

L'exemple de stratégie suivant applique deux restrictions basées sur des balises. Un utilisateur soumis à des restrictions en vertu de cette politique :

- Ne peut pas attribuer à une ressource la balise « env=prod » (dans l'exemple, voir la ligne `"aws:RequestTag/env" : "prod"`)
- Ne peut pas modifier une ressource qui a une balise existante « env=prod » ou y accéder (dans l'exemple, voir la ligne `"aws:ResourceTag/env" : "prod"`).

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Deny",
            "Action": [
                "iotevents:CreateDetectorModel",
                "iotevents:CreateAlarmModel",
                "iotevents:CreateInput",
                "iotevents:TagResource"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "aws:RequestTag/env": "prod"
                }
            }
        },
        {
            "Effect": "Deny",
            "Action": [
                "iotevents:DescribeDetectorModel",
                "iotevents:DescribeAlarmModel",
                "iotevents:UpdateDetectorModel",
                "iotevents:UpdateAlarmModel",
                "iotevents:DeleteDetectorModel",
                "iotevents:DeleteAlarmModel",
                "iotevents:ListDetectorModelVersions",
                "iotevents:ListAlarmModelVersions",
                "iotevents:UpdateInput",
                "iotevents:DescribeInput",
                "iotevents:DeleteInput",
                "iotevents:ListTagsForResource",
                "iotevents:TagResource",
                "iotevents:UntagResource",
                "iotevents:UpdateInputRouting"
            ],
            "Resource": "*",
            "Condition": {
                "StringLike": {
                    "aws:ResourceTag/env": "prod"
```

```
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "iotevents:*"
            ],
            "Resource": "*"
        }
    ]
}
```

Vous pouvez également spécifier plusieurs valeurs de balise pour une clé de balise donnée en les incluant dans une liste, comme suit.

```
        "StringEquals" : {
          "aws:ResourceTag/env" : ["dev", "test"]
        }
```

> **Note**
>
> Si vous autorisez ou refusez à des utilisateurs l'accès à des ressources en fonction de balises, vous devez envisager de refuser de manière explicite la possibilité pour les utilisateurs d'ajouter ces balises ou de les supprimer des mêmes ressources. Sinon, il sera possible pour un utilisateur de contourner vos restrictions et d'obtenir l'accès à une ressource en modifiant ses balises.

Résolution des problèmes AWS IoT Events

Ce guide de dépannage fournit des solutions aux problèmes courants que vous pouvez rencontrer lors de l'utilisation AWS IoT Events. Parcourez les rubriques pour identifier et résoudre les problèmes liés à la détection d'événements, à l'accès aux données, aux autorisations, aux intégrations de services, à la configuration des appareils, etc. Ce guide contient des conseils de dépannage concernant la AWS IoT Events console, l'API, la CLI, les erreurs, la latence et les intégrations. Il vise à résoudre rapidement vos problèmes afin que vous puissiez créer des applications basées sur les événements fiables et évolutives.

Rubriques

- AWS IoT Events Problèmes courants et solutions
- Résolution des problèmes liés à un modèle de détecteur en exécutant des analyses dans AWS IoT Events

AWS IoT Events Problèmes courants et solutions

Consultez la section suivante pour résoudre les erreurs et trouver des solutions possibles pour résoudre les problèmes liés à AWS IoT Events.

Erreurs

- Erreurs de création du modèle de détecteur
- Mises à jour depuis un modèle de détecteur supprimé
- Défaillance du déclencheur d'action (en cas de respect d'une condition)
- Défaillance du déclencheur de l'action (en cas de dépassement d'un seuil)
- Utilisation incorrecte de l'état
- Message de connexion
- InvalidRequestException message
- action.setTimerErreurs Amazon CloudWatch Logs
- Erreurs de CloudWatch charge utile Amazon
- Types de données incompatibles
- Impossible d'envoyer le message à AWS IoT Events

Erreurs de création du modèle de détecteur

Je reçois des erreurs lorsque je tente de créer un modèle de détecteur.

Solution

Lorsque vous créez un modèle de détecteur, vous devez tenir compte des limites suivantes.

- Une seule action est autorisée dans chaque `action` champ.
- Le `condition` est requis pour`transitionEvents`. C'est facultatif pour `OnEnter``OnInput`, et les `OnExit` événements.
- Si le `condition` champ est vide, le résultat évalué de l'expression de condition est équivalent à`true`.
- Le résultat évalué de l'expression de condition doit être une valeur booléenne. Si le résultat n'est pas une valeur booléenne, il est équivalent à la valeur `nextState` spécifiée dans l'`actions`événement `false` et ne déclenche pas la transition vers celle-ci.

Pour de plus amples informations, veuillez consulter AWS IoT Events restrictions et limites du modèle de détecteur.

Mises à jour depuis un modèle de détecteur supprimé

J'ai mis à jour ou supprimé un modèle de détecteur il y a quelques minutes, mais je reçois toujours des mises à jour de l'état de l'ancien modèle de détecteur par le biais de messages MQTT ou d'alertes SNS.

Solution

Si vous mettez à jour, supprimez ou recréez un modèle de détecteur (voir UpdateDetectorModel), il y a un délai avant que toutes les instances de détecteur soient supprimées et que le nouveau modèle soit utilisé. Pendant ce temps, les entrées peuvent continuer à être traitées par les instances de la version précédente du modèle de détecteur. Il est possible que vous continuiez à recevoir les alertes définies par le modèle de détecteur précédent. Patientez au moins sept minutes avant de vérifier à nouveau la mise à jour ou de signaler une erreur.

Défaillance du déclencheur d'action (en cas de respect d'une condition)

Le détecteur ne parvient pas à déclencher une action ou à passer à un nouvel état lorsque la condition est remplie.

Solution

Vérifiez que le résultat évalué de l'expression conditionnelle du détecteur est une valeur booléenne. Si le résultat n'est pas une valeur booléenne, il est équivalent à la valeur `nextState` spécifiée dans l'`action`événement `false` et ne déclenche pas la transition vers celle-ci. Pour plus d'informations, consultez Syntaxe des expressions conditionnelles.

Défaillance du déclencheur de l'action (en cas de dépassement d'un seuil)

Le détecteur ne déclenche pas d'action ou de transition d'événement lorsque la variable d'une expression conditionnelle atteint une valeur spécifiée.

Solution

Si vous effectuez `setVariable` une mise à jour pour `onInput`ou`onEnter`, ou`onExit`, la nouvelle valeur n'est pas utilisée lors de l'évaluation d'une valeur `condition` au cours du cycle de traitement en cours. Au lieu de cela, la valeur d'origine est utilisée jusqu'à ce que le cycle en cours soit terminé. Vous pouvez modifier ce comportement en définissant le `evaluationMethod` paramètre dans la définition du modèle de détecteur. Lorsque `evaluationMethod` ce paramètre est défini sur`SERIAL`, les variables sont mises à jour et les conditions des événements sont évaluées dans l'ordre dans lequel les événements sont définis. Lorsque `evaluationMethod` ce paramètre est défini sur `BATCH` (valeur par défaut), les variables sont mises à jour et les événements ne sont exécutés qu'une fois que toutes les conditions de l'événement ont été évaluées.

Utilisation incorrecte de l'état

Le détecteur entre dans les mauvais états lorsque j'essaie d'envoyer des messages aux entrées en utilisant`BatchPutMessage`.

Solution

Si vous avez l'BatchPutMessagehabitude d'envoyer plusieurs messages aux entrées, l'ordre dans lequel les messages ou les entrées sont traités n'est pas garanti. Pour garantir la commande, envoyez les messages un par un et attendez à chaque fois `BatchPutMessage` de confirmer le succès.

Message de connexion

Je reçois un `('Connection aborted.', error(54, 'Connection reset by peer'))` message d'erreur lorsque je tente d'appeler ou d'invoquer une API.

Solution

Vérifiez qu'OpenSSL utilise le protocole TLS 1.1 ou une version ultérieure pour établir la connexion. Cela devrait être la valeur par défaut dans la plupart des distributions Linux ou dans les versions 7 et ultérieures de Windows. Les utilisateurs de macOS devront peut-être mettre à jour OpenSSL.

InvalidRequestException message

Je reçois InvalidRequestException quand j'essaie d'appeler `CreateDetectorModel` et `UpdateDetectorModel` APIs.

Solution

Vérifiez les points suivants pour résoudre le problème. Pour plus d'informations, consultez CreateDetectorModel et UpdateDetectorModel.

- Assurez-vous de ne pas utiliser `seconds` les deux `durationExpression` en même temps que les paramètres de`SetTimerAction`.
- Assurez-vous que l'expression de chaîne pour `durationExpression` est valide. L'expression sous forme de chaîne peut contenir des nombres, des variables (`$variable.<variable-name>`) ou des valeurs d'entrée (`$input.<input-name>.<path-to-datum>`).

`action.setTimer`Erreurs Amazon CloudWatch Logs

Vous pouvez configurer Amazon CloudWatch Logs pour surveiller les instances AWS IoT Events de modèles de détecteurs. Les erreurs suivantes sont courantes générées par AWS IoT Events, lorsque vous utilisez`action.setTimer`.

- Erreur : votre expression de durée pour le temporisateur nommé n'a pas *`<timer-name>`* pu être évaluée en nombre.

Solution

Assurez-vous que l'expression sous forme de chaîne pour `durationExpression` peut être convertie en nombre. Les autres types de données, tels que les données booléennes, ne sont pas autorisés.

- Erreur : le résultat évalué de votre expression de durée pour le temporisateur nommé *`<timer-name>`* est supérieur à 31622440. Pour garantir l'exactitude, assurez-vous que votre expression de durée fait référence à une valeur comprise entre 60-31622400.

Solution

Assurez-vous que la durée de votre chronomètre est inférieure ou égale à 31622400 secondes. Le résultat évalué de la durée est arrondi au nombre entier inférieur le plus proche.

- Erreur : le résultat évalué de votre expression de durée pour le temporisateur nommé *<timer-name>* est inférieur à 60. Pour garantir l'exactitude, assurez-vous que votre expression de durée fait référence à une valeur comprise entre 60-31622400.

Solution

Assurez-vous que la durée de votre chronomètre est supérieure ou égale à 60 secondes. Le résultat évalué de la durée est arrondi au nombre entier inférieur le plus proche.

- Erreur : votre expression de durée pour le temporisateur nommé n'a pas *<timer-name>* pu être évaluée. Vérifiez les noms des variables, les noms des entrées et les chemins d'accès aux données pour vous assurer que vous faites référence aux variables et entrées existantes.

Solution

Assurez-vous que votre expression sous forme de chaîne fait référence aux variables et entrées existantes. L'expression sous forme de chaîne peut contenir des nombres, des variables (`$variable.`*`variable-name`*) et des valeurs d'entrée (`$input.`*`input-name`*`.`*`path-to-datum`*).

- Erreur : Impossible de définir le minuteur nommé`<timer-name>`. Vérifiez votre expression de durée, puis réessayez.

Solution

Consultez l'SetTimerActionaction pour vous assurer que vous avez spécifié les bons paramètres, puis réglez à nouveau le minuteur.

Pour plus d'informations, consultez Activer la CloudWatch journalisation Amazon lors du développement de modèles AWS IoT Events de détecteurs.

Erreurs de CloudWatch charge utile Amazon

Vous pouvez configurer Amazon CloudWatch Logs pour surveiller les instances AWS IoT Events de modèles de détecteurs. Vous trouverez ci-dessous les erreurs et les avertissements courants générés par AWS IoT Events, lorsque vous configurez la charge utile de l'action.

- Erreur : nous n'avons pas pu évaluer votre expression pour l'action. Assurez-vous que les noms des variables, les noms d'entrée et les chemins d'accès aux données font référence aux variables et aux valeurs d'entrée existantes. Vérifiez également que la taille de la charge utile est inférieure à 1 Ko, la taille maximale autorisée d'une charge utile.

Solution

Assurez-vous de saisir les noms de variables, les noms d'entrée et les chemins d'accès aux données corrects. Ce message d'erreur peut également s'afficher si la charge utile de l'action est supérieure à 1 Ko.

- Erreur : nous n'avons pas pu analyser votre expression de contenu pour la charge utile de. *<action-type>* Entrez une expression de contenu avec la syntaxe correcte.

Solution

L'expression de contenu peut contenir des chaînes (`'`*`string`*`'`), des variables (`$variable.`*`variable-name`*), des valeurs d'entrée (`$input.`*`input-name`*`.`*`path-to-datum`*), des concaténations de chaînes et des chaînes contenant. `${}`

- Erreur : votre expression de charge utile {*expression*} n'est pas valide. Le type de charge utile défini est JSON. Vous devez donc spécifier une expression qui AWS IoT Events serait évaluée en chaîne.

Solution

Si le type de charge utile spécifié est JSON, AWS IoT Events vérifiez d'abord si le service peut évaluer votre expression sous forme de chaîne. Le résultat évalué ne peut pas être un booléen ou un nombre. Si la validation échoue, il se peut que vous receviez cette erreur.

- Avertissement : L'action a été exécutée, mais nous n'avons pas pu évaluer votre expression de contenu pour déterminer si la charge utile de l'action était au format JSON valide. Le type de charge utile défini est JSON.

Solution

Assurez-vous qu'il AWS IoT Events peut évaluer votre expression de contenu pour que la charge utile de l'action soit au format JSON valide, si vous définissez le type de charge utile comme. `JSON` AWS IoT Events exécute l'action même s'il n'est pas AWS IoT Events possible d'évaluer l'expression de contenu en JSON valide.

Pour plus d'informations, consultez Activer la CloudWatch journalisation Amazon lors du développement de modèles AWS IoT Events de détecteurs.

Types de données incompatibles

Message : types de données incompatibles [`<inferred-types>`] trouvés `<reference>` dans l'expression suivante : `<expression>`

Solution

Ce message d'erreur peut s'afficher pour l'une des raisons suivantes :

- Les résultats évalués de vos références ne sont pas compatibles avec les autres opérandes de vos expressions.
- Le type de l'argument transmis à une fonction n'est pas pris en charge.

Lorsque vous utilisez des références dans des expressions, vérifiez les points suivants :

- Lorsque vous utilisez une référence comme opérande avec un ou plusieurs opérateurs, assurez-vous que tous les types de données auxquels vous faites référence sont compatibles.

  Par exemple, dans l'expression suivante, le nombre entier 2 est un opérande des && opérateurs == et. Pour garantir la compatibilité des opérandes, `$variable.testVariable + 1` ils `$variable.testVariable` doivent faire référence à un entier ou à un nombre décimal.

  De plus, le nombre entier 1 est un opérande de l'+opérateur. Par conséquent, `$variable.testVariable` doit faire référence à un entier ou à un nombre décimal.

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

- Lorsque vous utilisez une référence comme argument transmis à une fonction, assurez-vous que celle-ci prend en charge les types de données auxquels vous faites référence.

  Par exemple, la `timeout("time-name")` fonction suivante nécessite une chaîne avec des guillemets comme argument. Si vous utilisez une référence pour la *`timer-name`* valeur, vous devez référencer une chaîne entre guillemets doubles.

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

> **Note**
>
> Pour la `convert(type, expression)` fonction, si vous utilisez une référence pour la *type* valeur, le résultat évalué de votre référence doit être `StringDecimal`, ou`Boolean`.

Pour de plus amples informations, veuillez consulter AWS IoT Events référence pour les entrées et les variables dans les expressions.

Impossible d'envoyer le message à AWS IoT Events

Message : Impossible d'envoyer le message à Iot Events

Solution

Cette erreur peut se produire pour les raisons suivantes :

- La charge utile du message d'entrée ne contient pas le`Input attribute Key`.
- Le n'`Input attribute Key`est pas dans le même chemin JSON que celui spécifié dans la définition d'entrée.
- Le message d'entrée ne correspond pas au schéma tel que défini dans l' AWS IoT Events entrée.

> **Note**
>
> L'ingestion de données provenant d'autres services échouera également.

Example

Par exemple AWS IoT Core, dans, la AWS IoT règle échouera avec le message suivant `Verify the Input Attribute key.`

Pour résoudre ce problème, assurez-vous que le schéma du message de charge utile d'entrée est conforme à la définition AWS IoT Events d'entrée et que l'`Input attribute Key`emplacement correspond. Pour plus d'informations, consultez Créez une entrée pour les modèles dans AWS IoT Events la section pour savoir comment définir les AWS IoT Events entrées.

Résolution des problèmes liés à un modèle de détecteur en exécutant des analyses dans AWS IoT Events

AWS IoT Events peut analyser votre modèle de détecteur et générer des résultats d'analyse sans envoyer de données d'entrée à votre modèle de détecteur. AWS IoT Events effectue une série d'analyses décrites dans cette section pour vérifier le modèle de votre détecteur. Cette solution de dépannage avancée résume également les informations de diagnostic, notamment le niveau de gravité et l'emplacement, afin que vous puissiez rapidement détecter et résoudre les problèmes potentiels liés à votre modèle de détecteur. Pour plus d'informations sur les types d'erreurs de diagnostic et les messages relatifs à votre modèle de détecteur, consultezAnalyse du modèle de détecteur et informations diagnostiques pour AWS IoT Events.

Vous pouvez utiliser la AWS IoT Events console, l'API AWS Command Line Interface (AWS CLI) ou le AWS SDK pour afficher les messages d'erreur de diagnostic issus de l'analyse de votre modèle de détecteur.

> **Note**
>
> - Vous devez corriger toutes les erreurs avant de pouvoir publier votre modèle de détecteur.
> - Nous vous recommandons de consulter les avertissements et de prendre les mesures nécessaires avant d'utiliser votre modèle de détecteur dans des environnements de production. Dans le cas contraire, le modèle de détecteur risque de ne pas fonctionner comme prévu.
> - Vous pouvez avoir jusqu'à 10 analyses dans le `RUNNING` statut en même temps.

Pour savoir comment analyser votre modèle de détecteur, consultez Analyser un modèle de détecteur pour AWS IoT Events (Console) ouAnalyser un modèle de détecteur dans AWS IoT Events (AWS CLI).

Rubriques

- Analyse du modèle de détecteur et informations diagnostiques pour AWS IoT Events
- Analyser un modèle de détecteur pour AWS IoT Events (Console)
- Analyser un modèle de détecteur dans AWS IoT Events (AWS CLI)

Analyse du modèle de détecteur et informations diagnostiques pour AWS IoT Events

Les analyses du modèle de détecteur permettent de recueillir les informations de diagnostic suivantes :

- Niveau : niveau de gravité du résultat de l'analyse. Selon le niveau de gravité, les résultats de l'analyse se répartissent en trois catégories générales :
  - Information (`INFO`) — Un résultat d'information vous indique l'existence d'un champ significatif dans votre modèle de détecteur. Ce type de résultat ne nécessite généralement pas d'action immédiate.
  - Avertissement (`WARNING`) — Un résultat d'avertissement attire particulièrement l'attention sur les champs susceptibles de poser des problèmes pour votre modèle de détecteur. Nous vous recommandons de consulter les avertissements et de prendre les mesures nécessaires avant d'utiliser votre modèle de détecteur dans des environnements de production. Dans le cas contraire, le modèle de détecteur risque de ne pas fonctionner comme prévu.
  - Erreur (`ERROR`) — Un résultat d'erreur vous signale un problème détecté dans votre modèle de détecteur. AWS IoT Events effectue automatiquement cet ensemble d'analyses lorsque vous essayez de publier le modèle du détecteur. Vous devez corriger toutes les erreurs avant de pouvoir publier le modèle du détecteur.
- Emplacement : contient des informations que vous pouvez utiliser pour localiser le champ dans votre modèle de détecteur auquel le résultat de l'analyse fait référence. Un emplacement inclut généralement le nom de l'État, le nom de l'événement de transition, le nom de l'événement et l'expression (par exemple,`in state TemperatureCheck in onEnter in event Init in action setVariable`).
- Type : type du résultat de l'analyse. Les types d'analyse appartiennent aux catégories suivantes :
  - `supported-actions`— AWS IoT Events peut invoquer des actions lorsqu'un événement spécifique ou un événement de transition est détecté. Vous pouvez définir des actions intégrées pour utiliser un temporisateur ou définir une variable, ou envoyer des données à d'autres AWS services. Vous devez spécifier des actions qui fonctionnent avec d'autres AWS services dans une AWS région où les AWS services sont disponibles.
  - `service-limits`— Les quotas de service, également appelés limites, sont le nombre maximum ou minimum de ressources de service ou d'opérations pour votre AWS compte. Sauf indication contraire, chaque quota est spécifique à une région. En fonction des besoins de votre entreprise, vous pouvez mettre à jour le modèle de votre détecteur pour éviter de rencontrer des

limites ou demander une augmentation de quota. Vous pouvez demander des augmentations pour certains quotas, et d'autres quotas ne peuvent pas être augmentés. Pour de plus amples informations, veuillez consulter Quotas .

- **`structure`**— Le modèle de détecteur doit comporter tous les composants requis, tels que les états, et suivre une structure qui le AWS IoT Events supporte. Un modèle de détecteur doit avoir au moins un état et une condition qui évaluent les données d'entrée entrantes pour détecter des événements significatifs. Lorsqu'un événement est détecté, le modèle de détecteur passe à l'état suivant et peut invoquer des actions. Ces événements sont appelés événements de transition. Un événement de transition doit indiquer l'état suivant à entrer.
- **`expression-syntax`**— AWS IoT Events propose plusieurs méthodes pour spécifier des valeurs lors de la création et de la mise à jour de modèles de détecteurs. Vous pouvez utiliser des littéraux, des opérateurs, des fonctions, des références et des modèles de substitution dans les expressions. Vous pouvez utiliser des expressions pour spécifier des valeurs littérales ou AWS IoT Events évaluer les expressions avant de spécifier des valeurs particulières. Votre expression doit respecter la syntaxe requise. Pour de plus amples informations, veuillez consulter Expressions pour filtrer, transformer et traiter les données d'événements.

  Les expressions du modèle de détecteur AWS IoT Events peuvent faire référence à des données ou à une ressource spécifiques.

  - **`data-type`**— AWS IoT Events prend en charge les types de données entiers, décimaux, chaînes et booléens. S'il est AWS IoT Events possible de convertir automatiquement les données d'un type de données en un autre lors de l'évaluation de l'expression, ces types de données sont compatibles.

    > **Note**
    > - Les types de données entiers et décimaux sont les seuls types de données compatibles pris en charge par AWS IoT Events.
    > - AWS IoT Events ne peut pas évaluer les expressions arithmétiques car il AWS IoT Events est impossible de convertir un entier en chaîne.

  - **`referenced-data`**— Vous devez définir les données référencées dans votre modèle de détecteur avant de pouvoir les utiliser. Par exemple, si vous souhaitez envoyer des données vers une table DynamoDB, vous devez définir une variable qui fait référence au nom de la table avant de pouvoir utiliser la variable dans une expression (). `$variable.TableName`

- **`referenced-resource`**— Les ressources utilisées par le modèle de détecteur doivent être disponibles. Vous devez définir les ressources avant de pouvoir les utiliser. Par exemple, vous souhaitez créer un modèle de détecteur pour surveiller la température d'une serre. Vous devez définir une entrée (`$input.TemperatureInput`) pour acheminer les données de température entrantes vers votre modèle de détecteur avant de pouvoir utiliser le `$input.TemperatureInput.sensorData.temperature` pour référencer la température.

Consultez la section suivante pour résoudre les erreurs et trouver des solutions possibles à partir de l'analyse de votre modèle de détecteur.

Résoudre les erreurs de modèle de détecteur dans AWS IoT Events

Les types d'erreurs décrits ci-dessus fournissent des informations de diagnostic sur un modèle de détecteur et correspondent aux messages que vous pouvez récupérer. Utilisez ces messages et les solutions proposées pour résoudre les erreurs liées à votre modèle de détecteur.

Messages et solutions

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

`Location`

Un résultat d'analyse contenant des informations sur`Location`, correspond au message d'erreur suivant :

- Message : contient des informations supplémentaires sur le résultat de l'analyse. Il peut s'agir d'un message d'information, d'avertissement ou d'erreur.

  Solution : ce message d'erreur peut s'afficher si vous avez spécifié une action qui n'est AWS IoT Events actuellement pas prise en charge. Pour obtenir la liste des actions prises en charge,

consultezActions prises en charge pour recevoir des données et déclencher des actions dans AWS IoT Events.

`supported-actions`

Un résultat d'analyse contenant des informations sur`supported-actions`, correspond aux messages d'erreur suivants :

- Message : Type d'action non valide présent dans la définition de l'action :*`action-definition`*.

  Solution : ce message d'erreur peut s'afficher si vous avez spécifié une action qui n'est AWS IoT Events actuellement pas prise en charge. Pour obtenir la liste des actions prises en charge, consultezActions prises en charge pour recevoir des données et déclencher des actions dans AWS IoT Events.
- Message : DetectorModel la définition comporte une *`aws-service`* action, mais le *`aws-service`* service n'est pas pris en charge dans la région*`region-name`*.

  Solution : ce message d'erreur peut s'afficher si l'action que vous avez spécifiée est prise en charge par AWS IoT Events, mais qu'elle n'est pas disponible dans votre région actuelle. Cela peut se produire lorsque vous essayez d'envoyer des données à un AWS service qui n'est pas disponible dans la région. Vous devez également choisir la même région pour les deux AWS IoT Events et pour les AWS services que vous utilisez.

`service-limits`

Un résultat d'analyse contenant des informations sur`service-limits`, correspond aux messages d'erreur suivants :

- Message : L'expression de contenu autorisée dans la charge utile dépassait la limite d'*`content-expression-size`*octets dans l'état *`state-name`* de l'événement*`event-name`*.

  Solution : ce message d'erreur peut s'afficher si l'expression de contenu de votre charge utile d'action est supérieure à 1 024 octets. La taille de l'expression de contenu d'une charge utile peut atteindre 1 024 octets.
- Message : Le nombre d'états autorisés dans la définition du modèle de détecteur a dépassé la limite*`states-per-detector-model`*.

Solution : ce message d'erreur peut s'afficher si le modèle de votre détecteur comporte plus de 20 états. Un modèle de détecteur peut comporter jusqu'à 20 états.

- Message : La durée du chronomètre *`timer-name`* doit être d'au moins *`minimum-timer-duration`* quelques secondes.

  Solution : ce message d'erreur peut s'afficher si la durée du chronomètre est inférieure à 60 secondes. Nous recommandons que la durée d'un chronomètre soit comprise entre 60 et 31622400 secondes. Si vous spécifiez une expression pour la durée de votre chronomètre, le résultat évalué de l'expression de durée est arrondi au nombre entier inférieur le plus proche.

- Message : Le nombre d'actions autorisées par événement a dépassé la limite définie *`actions-per-event`* dans la définition du modèle de détecteur

  Solution : ce message d'erreur peut s'afficher si l'événement comporte plus de 10 actions. Vous pouvez avoir jusqu'à 10 actions pour chaque événement dans votre modèle de détecteur.

- Message : Le nombre d'événements de transition autorisés par état a dépassé la limite fixée *`transition-events-per-state`* dans la définition du modèle de détecteur.

  Solution : ce message d'erreur peut s'afficher si l'État compte plus de 20 événements de transition. Vous pouvez avoir jusqu'à 20 événements de transition pour chaque état de votre modèle de détecteur.

- Message : Le nombre d'événements autorisés par état a dépassé la limite fixée *`events-per-state`* dans la définition du modèle de détecteur

  Solution : ce message d'erreur peut s'afficher si l'État compte plus de 20 événements. Vous pouvez avoir jusqu'à 20 événements pour chaque état dans votre modèle de détecteur.

- Message : Le nombre maximum de modèles de détecteurs pouvant être associés à une seule entrée peut avoir atteint la limite. *`input-name`*L'entrée est utilisée dans les itinéraires des modèles de *`detector-models-per-input`* détecteurs.

  Solution : ce message d'avertissement peut s'afficher si vous essayez d'acheminer une entrée vers plus de 10 modèles de détecteurs. Vous pouvez associer jusqu'à 10 modèles de détecteurs différents à un seul modèle de détecteur.

`structure`

Un résultat d'analyse contenant des informations sur`structure`, correspond aux messages d'erreur suivants :

- Message : Les actions peuvent n'avoir qu'un seul type défini, mais une action contenant des *number-of-types* types a été trouvée. Répartissez-le en actions distinctes.

  Solution : ce message d'erreur peut s'afficher si vous avez spécifié deux actions ou plus dans un seul champ en utilisant des opérations d'API pour créer ou mettre à jour votre modèle de détecteur. Vous pouvez définir un tableau d'`Action`objets. Assurez-vous de définir chaque action comme un objet distinct.

- Message : La TransitionEvent *transition-event-name* transition vers un état *state-name* inexistant.

  Solution : ce message d'erreur peut s'afficher si vous AWS IoT Events ne trouvez pas le prochain état référencé par votre événement de transition. Assurez-vous que l'état suivant est défini et que vous avez saisi le nom d'état correct.

- Message : L'état DetectorModelDefinition avait un nom d'état partagé : état trouvé *state-name* avec *number-of-states* répétitions.

  Solution : ce message d'erreur peut s'afficher si vous utilisez le même nom pour un ou plusieurs états. Assurez-vous de donner un nom unique à chaque état de votre modèle de détecteur. Le nom de l'État doit comporter de 1 à 128 caractères. Caractères valides : a-z, A-Z, 0-9, _ (trait de soulignement) et - (tiret).

- Message : Les définitions ne initialStateName *initial-state-name* correspondaient pas à un état défini.

  Solution : ce message d'erreur peut s'afficher si le nom d'état initial est incorrect. Le modèle de détecteur reste dans l'état initial (démarrage) jusqu'à ce qu'une entrée arrive. Dès qu'une entrée arrive, le modèle de détecteur passe immédiatement à l'état suivant. Assurez-vous que le nom d'état initial est le nom d'un état défini et que vous entrez le nom correct.

- Message : La définition du modèle de détecteur doit utiliser au moins une entrée dans une condition.

  Solution : cette erreur peut s'afficher si vous n'avez pas spécifié d'entrée dans une condition. Vous devez utiliser au moins une entrée dans au moins une condition. Sinon, AWS IoT Events n'évalue pas les données entrantes.

- Message : une seule des secondes et DurationExpression peuvent être définies. SetTimer

  Solution : ce message d'erreur peut s'afficher si vous avez utilisé les deux `seconds` et `durationExpression` pour votre minuteur. Assurez-vous d'utiliser l'un `seconds` ou l'autre

`durationExpression` des paramètres de`SetTimerAction`. Pour plus d'informations, consultez SetTimerAction dans la Référence d'API AWS IoT Events .

- Message : Une action de votre modèle de détecteur est inaccessible. Vérifiez la condition qui déclenche l'action.

  Solution : Si une action de votre modèle de détecteur est inaccessible, la condition de l'événement est considérée comme fausse. Vérifiez la condition de l'événement contenant l'action pour vous assurer qu'elle est évaluée comme vraie. Lorsque la condition de l'événement devient vraie, l'action doit devenir accessible.

- Message : Un attribut d'entrée est en cours de lecture, mais cela peut être dû à l'expiration du délai d'expiration.

  Solution : La valeur d'un attribut d'entrée peut être lue lorsque l'une des situations suivantes se produit :

  - Une nouvelle valeur d'entrée a été reçue.
  - Lorsqu'une minuterie du détecteur a expiré.

  Pour garantir qu'un attribut d'entrée est évalué uniquement lorsque la nouvelle valeur de cette entrée est reçue, incluez un appel à la `triggerType("Message")` fonction dans votre condition comme suit :

  État d'origine évalué dans le modèle de détecteur :

  ```
  if ($input.HeartBeat.status == "OFFLINE")
  ```

  deviendrait similaire à ce qui suit :

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

  où un appel à la `triggerType("Message")` fonction intervient avant l'entrée initiale fournie dans la condition. En utilisant cette technique, la `triggerType("Message")` fonction sera évaluée comme vraie et satisfera à la condition de réception d'une nouvelle valeur d'entrée. Pour plus d'informations sur l'utilisation de cette `triggerType` fonction, recherchez `triggerType` dans la section Expressions du Guide du AWS IoT Events développeur

- Message : Un état de votre modèle de détecteur est inaccessible. Vérifiez la condition qui provoquera une transition vers l'état souhaité.

Solution : Si un état de votre modèle de détecteur est inaccessible, une condition provoquant une transition entrante vers cet état est considérée comme fausse. Vérifiez que les conditions des transitions entrantes vers cet état inaccessible dans votre modèle de détecteur sont vraies, afin que l'état souhaité puisse devenir accessible.

- Message : L'expiration d'un délai peut entraîner l'envoi d'un nombre inattendu de messages.

  Solution : pour éviter que votre modèle de détecteur n'entre dans un état infini d'envoi inattendu de messages en raison de l'expiration d'un délai, pensez à appeler la `triggerType("Message")` fonction, dans les conditions de votre modèle de détecteur, comme suit :

  État d'origine évalué dans le modèle de détecteur :

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

  serait transformé en une condition semblable à ce qui suit :

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

  où un appel à la `triggerType(“Message”)` fonction intervient avant l'entrée initiale fournie dans la condition.

  Cette modification empêche de lancer des actions de temporisation dans votre détecteur, empêchant ainsi l'envoi d'une boucle infinie de messages. Pour plus d'informations sur l'utilisation des actions du temporisateur dans votre détecteur, consultez la page Utilisation des actions intégrées du manuel du AWS IoT Events développeur

`expression-syntax`

Un résultat d'analyse contenant des informations sur`expression-syntax`, correspond aux messages d'erreur suivants :

- Message : Votre expression de charge utile {*`expression`*} n'est pas valide. Le type de charge utile défini est JSON. Vous devez donc spécifier une expression qui AWS IoT Events serait évaluée en chaîne.

Solution : Si le type de charge utile spécifié est JSON, vérifiez d' AWS IoT Events abord si le service peut évaluer votre expression sous forme de chaîne. Le résultat évalué ne peut pas être un booléen ou un nombre. Si la validation échoue, il se peut que vous receviez cette erreur.

- Message : `SetVariableAction.value` doit être une expression. Impossible d'analyser la valeur « » *`variable-value`*

  Solution : Vous pouvez l'utiliser `SetVariableAction` pour définir une variable avec un `name` et`value`. Il `value` peut s'agir d'une chaîne, d'un nombre ou d'une valeur booléenne. Vous pouvez également spécifier une expression pour`value`. Pour plus d'informations SetVariableAction, reportez-vous à la section Référence des AWS IoT Events API.

- Message : Nous n'avons pas pu analyser l'expression des attributs (*`attribute-name`*) pour l'action DynamoDB. Entrez une expression avec la syntaxe correcte.

  Solution : Vous devez utiliser des expressions pour tous les paramètres des modèles de substitution`DynamoDBAction`. Pour plus d'informations, consultez Dynamo DBAction dans le manuel de référence des AWS IoT Events API.

- Message : Nous n'avons pas pu analyser l'expression du TableName pour l'action Dynamo. DBv2 Entrez une expression avec la syntaxe correcte.

  Solution : L'`tableName`entrée `DynamoDBv2Action` doit être une chaîne. Vous devez utiliser une expression pour`tableName`. Les expressions acceptent les littéraux, les opérateurs, les fonctions, les références et les modèles de substitution. Pour plus d'informations, consultez Dynamo DBv2 Action dans le Guide de référence de l'AWS IoT Events API.

- Message : Nous n'avons pas pu évaluer votre expression au format JSON valide. L'DBv2 action Dynamo prend uniquement en charge le type de charge utile JSON.

  Solution : le type de charge utile pour `DynamoDBv2` doit être JSON. Assurez-vous qu'il AWS IoT Events peut évaluer votre expression de contenu pour que la charge utile soit conforme à un JSON valide. Pour plus d'informations, consultez Dynamo DBv2 Action dans le Guide de référence des AWS IoT Events API.

- Message : Nous n'avons pas pu analyser votre expression de contenu pour la charge utile de. *`action-type`* Entrez une expression de contenu avec la syntaxe correcte.

  Solution : L'expression de contenu peut contenir des chaînes (« *`string`* »), des variables ($variable). *`variable-name`*), valeurs d'entrée ($input. *`input-name`*. *`path-to-datum`*), les concaténations de chaînes et les chaînes contenant. `${}`

- Message : Les charges utiles personnalisées ne doivent pas être vides.

  Solution : ce message d'erreur peut s'afficher si vous avez choisi une charge utile personnalisée pour votre action et que vous n'avez pas saisi d'expression de contenu dans la AWS IoT Events console. Si vous choisissez Charge utile personnalisée, vous devez saisir une expression de contenu sous Charge utile personnalisée. Pour plus d'informations, consultez la section Charge utile dans la référence de l'AWS IoT Events API.

- Message : Impossible d'analyser l'expression de durée « *`duration-expression`* » pour le temporisateur « *`timer-name`* ».

  Solution : Le résultat évalué de votre expression de durée pour le temporisateur doit être une valeur comprise entre 60 et 31622400. Le résultat évalué de la durée est arrondi au nombre entier inférieur le plus proche.

- Message : Impossible d'analyser l'expression « *`expression`* » pour *`action-name`*

  Solution : vous pouvez recevoir ce message si la syntaxe de l'expression de l'action spécifiée est incorrecte. Assurez-vous de saisir une expression dont la syntaxe est correcte. Pour de plus amples informations, veuillez consulter Syntaxe permettant de filtrer les données de l'appareil et de définir des actions dans AWS IoT Events.

- Message : Votre formulaire *`fieldName`* `IotSitewiseAction` n'a pas pu être analysé. Vous devez utiliser une syntaxe correcte dans votre expression.

  Solution : cette erreur peut s'afficher si vous AWS IoT Events ne parvenez pas à analyser votre *`fieldName`* for`IotSitewiseAction`. Assurez-vous qu'il *`fieldName`* utilise une expression qui AWS IoT Events peut être analysée. Pour plus d'informations, consultez IotSiteWiseAction dans la Référence d'API AWS IoT Events .

`data-type`

Un résultat d'analyse contenant des informations sur`data-type`, correspond aux messages d'erreur suivants :

- Message : L'expression de durée *`duration-expression`* pour le temporisateur n'*`timer-name`*est pas valide, elle doit renvoyer un nombre.

  Solution : ce message d'erreur peut s'afficher si vous AWS IoT Events ne parvenez pas à évaluer l'expression de durée de votre minuteur à un nombre. Assurez-vous que vous

`durationExpression` pouvez être converti en nombre. Les autres types de données, tels que les données booléennes, ne sont pas pris en charge.

- Message : L'expression n'*`condition-expression`*est pas une expression de condition valide.

  Solution : ce message d'erreur peut s'afficher si vous AWS IoT Events ne parvenez pas à évaluer votre valeur `condition-expression` à une valeur booléenne. La valeur booléenne doit être soit`TRUE`. `FALSE` Assurez-vous que votre expression de condition peut être convertie en valeur booléenne. Si le résultat n'est pas une valeur booléenne, il est équivalent aux actions spécifiées dans l'événement `FALSE` et n'appelle pas la transition vers la valeur `nextState` spécifiée dans l'événement.

- Message : types de données incompatibles [*`inferred-types`*] trouvés *`reference`* dans l'expression suivante : *`expression`*

  Solution : Toutes les expressions pour le même attribut ou variable d'entrée dans le modèle de détecteur doivent faire référence au même type de données.

  Utilisez les informations suivantes pour résoudre le problème :

  - Lorsque vous utilisez une référence comme opérande avec un ou plusieurs opérateurs, assurez-vous que tous les types de données auxquels vous faites référence sont compatibles.

    Par exemple, dans l'expression suivante, le nombre entier 2 est un opérande des && opérateurs == et. Pour garantir la compatibilité des opérandes, `$variable.testVariable + 1` ils `$variable.testVariable` doivent faire référence à un entier ou à un nombre décimal.

    De plus, le nombre entier 1 est un opérande de l'+opérateur. Par conséquent, `$variable.testVariable` doit faire référence à un entier ou à un nombre décimal.

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

  - Lorsque vous utilisez une référence comme argument transmis à une fonction, assurez-vous que celle-ci prend en charge les types de données auxquels vous faites référence.

    Par exemple, la `timeout("`*`time-name`*`")` fonction suivante nécessite une chaîne avec des guillemets comme argument. Si vous utilisez une référence pour la *`timer-name`* valeur, vous devez référencer une chaîne entre guillemets doubles.

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

> **Note**
>
> Pour la `convert(type, expression)` fonction, si vous utilisez une référence pour la *type* valeur, le résultat évalué de votre référence doit être `StringDecimal`, ou`Boolean`.

Pour de plus amples informations, veuillez consulter AWS IoT Events référence pour les entrées et les variables dans les expressions.

- Message : types de données incompatibles [*inferred-types*] utilisés avec*reference*. Cela peut entraîner une erreur d'exécution.

  Solution : ce message d'avertissement peut s'afficher si deux expressions pour le même attribut ou variable d'entrée font référence à deux types de données. Assurez-vous que vos expressions pour le même attribut ou variable d'entrée font référence au même type de données dans le modèle de détecteur.

- Message : Les types de données [*inferred-types*] que vous avez saisis pour l'opérateur [*operator*] ne sont pas compatibles avec l'expression suivante : « *expression* »

  Solution : ce message d'erreur peut s'afficher si votre expression combine des types de données incompatibles avec un opérateur spécifié. Par exemple, dans l'expression suivante, l'opérateur + est compatible avec les types de données Integer, Decimal et String, mais pas avec les opérandes de type booléen.

  ```
  true + false
  ```

  Vous devez vous assurer que les types de données que vous utilisez avec un opérateur sont compatibles.

- Message : Les types de données [*inferred-types*] trouvés *input-attribute* ne sont pas compatibles et peuvent entraîner une erreur d'exécution.

  Solution : ce message d'erreur peut s'afficher si deux expressions pour le même attribut d'entrée font référence à deux types `OnEnterLifecycle` de données soit pour un état, soit pour `OnInputLifecycle` les `OnExitLifecycle` deux. Assurez-vous que vos expressions dans `OnEnterLifecycle` (ou dans `OnInputLifecycle` les deux`OnExitLifecycle`) font référence au même type de données pour chaque état de votre modèle de détecteur.

- Message : L'expression de charge utile [*expression*] n'est pas valide. Spécifiez une expression qui serait évaluée en chaîne au moment de l'exécution, car le type de charge utile est au format JSON.

  Solution : cette erreur peut s'afficher si le type de charge utile que vous avez spécifié est JSON, mais que AWS IoT Events vous ne pouvez pas évaluer son expression sous forme de chaîne. Assurez-vous que le résultat évalué est une chaîne et non un booléen ou un nombre.

- Message : Votre expression interpolée {*interpolated-expression*} doit être évaluée à un entier ou à une valeur booléenne au moment de l'exécution. Sinon, votre expression de charge utile {*payload-expression*} ne pourra pas être analysée au moment de l'exécution en tant que JSON valide.

  Solution : ce message d'erreur peut s'afficher si vous AWS IoT Events ne parvenez pas à évaluer votre expression interpolée à un entier ou à une valeur booléenne. Assurez-vous que votre expression interpolée peut être convertie en entier ou en valeur booléenne, car les autres types de données, tels que les chaînes, ne sont pas pris en charge.

- Message : Le type d'expression dans le `IotSitewiseAction` champ *expression* est défini en tant que type *defined-type* et déduit en tant que type*inferred-type*. Le type défini et le type inféré doivent être identiques.

  Solution : ce message d'erreur peut s'afficher si votre expression dans le `propertyValue` de `IotSitewiseAction` possède un type de données défini différemment du type de données déduit par AWS IoT Events. Assurez-vous d'utiliser le même type de données pour toutes les instances de cette expression dans votre modèle de détecteur.

- Message : Les types de données [*inferred-types*] utilisés pour `setTimer` l'action ne sont pas évalués à `Integer` pour l'expression suivante : *expression*

  Solution : ce message d'erreur peut s'afficher si le type de données déduit pour votre expression de durée n'est pas un entier ou un décimal. Assurez-vous que vous `durationExpression` pouvez être converti en nombre. Les autres types de données, tels que Boolean et String, ne sont pas pris en charge.

- Message : Les types de données [*inferred-types*] utilisés avec les opérandes de l'opérateur de comparaison [*operator*] ne sont pas compatibles dans l'expression suivante : *expression*

  Solution : Les types de données déduits pour les opérandes de l'*operator*expression conditionnelle (*expression*) de votre modèle de détecteur ne correspondent pas. Les opérandes

doivent être utilisés avec les types de données correspondants dans toutes les autres parties de votre modèle de détecteur.

> **Tip**
>
> Vous pouvez l'utiliser `convert` pour modifier le type de données d'une expression dans votre modèle de détecteur. Pour de plus amples informations, veuillez consulter Fonctions à utiliser dans les AWS IoT Events expressions.

`referenced-data`

Un résultat d'analyse contenant des informations sur`referenced-data`, correspond aux messages d'erreur suivants :

- Message : Minuteur cassé détecté : le temporisateur *`timer-name`* est utilisé dans une expression mais n'est jamais défini.

  Solution : ce message d'erreur peut s'afficher si vous utilisez une minuterie qui n'est pas réglée. Vous devez définir un temporisateur avant de l'utiliser dans une expression. Assurez-vous également de saisir le nom du chronomètre correct.
- Message : Variable défectueuse détectée : la variable *`variable-name`* est utilisée dans une expression mais n'est jamais définie.

  Solution : ce message d'erreur peut s'afficher si vous utilisez une variable non définie. Vous devez définir une variable avant de l'utiliser dans une expression. Assurez-vous également de saisir le nom de variable correct.
- Message : Variable défectueuse détectée : une variable est utilisée dans une expression avant d'être définie sur une valeur.

  Solution : Chaque variable doit être affectée à une valeur avant de pouvoir être évaluée dans une expression. Définissez la valeur de la variable avant chaque utilisation afin que sa valeur puisse être récupérée. Assurez-vous également de saisir le nom de variable correct.

`referenced-resource`

Un résultat d'analyse contenant des informations sur`referenced-resource`, correspond aux messages d'erreur suivants :

- Message : La définition du modèle de détecteur contient une référence à une entrée qui n'existe pas.

  Solution : ce message d'erreur peut s'afficher si vous utilisez des expressions pour faire référence à une entrée qui n'existe pas. Assurez-vous que votre expression fait référence à une entrée existante et entrez le nom d'entrée correct. Si vous n'avez pas d'entrée, créez-en une d'abord.
- Message : La définition du modèle de détecteur contient des informations non valides InputName : *`input-name`*

  Solution : ce message d'erreur peut s'afficher si le modèle de votre détecteur contient un nom d'entrée non valide. Assurez-vous d'avoir saisi le bon nom d'entrée. Le nom d'entrée doit comporter de 1 à 128 caractères. Caractères valides : a-z, A-Z, 0-9, _ (trait de soulignement) et - (tiret).

Analyser un modèle de détecteur pour AWS IoT Events (Console)

AWS IoT Events vous permet de surveiller et de réagir aux données de l'IoT en détectant les événements et en déclenchant des actions avec l' AWS IoT Events API. Les étapes suivantes utilisent la AWS IoT Events console pour analyser un modèle de détecteur.

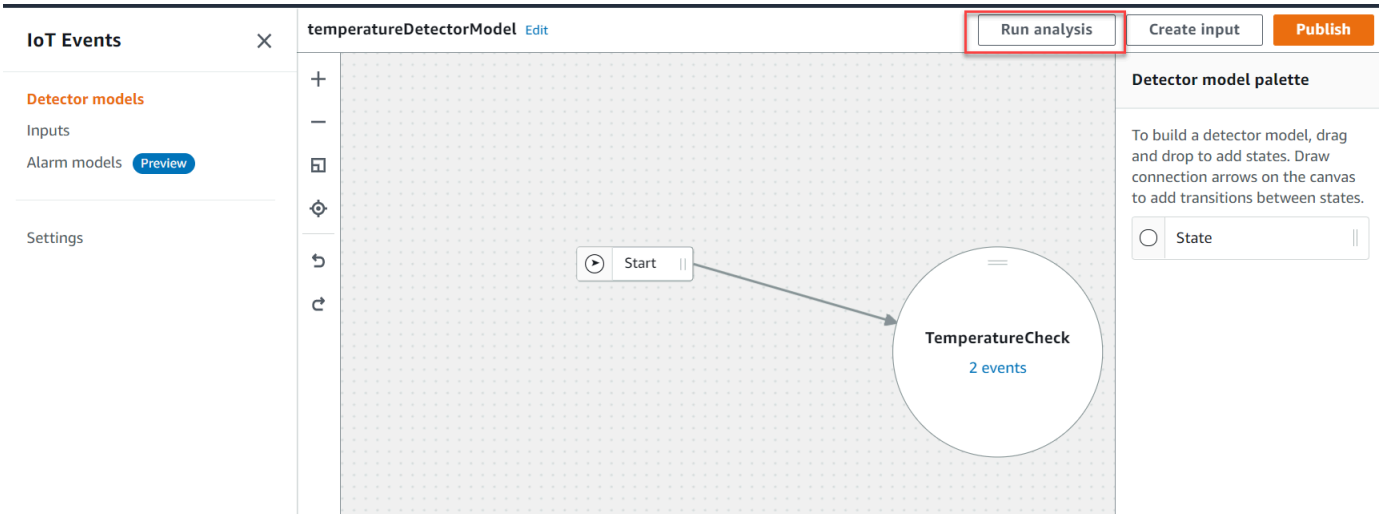
> **Note**
>
> Après avoir AWS IoT Events commencé à analyser votre modèle de détecteur, vous avez jusqu'à 24 heures pour récupérer les résultats de l'analyse.

L'analyse d'un modèle de détecteur peut vous aider à optimiser vos modèles, à identifier les problèmes potentiels et à garantir qu'ils fonctionnent comme prévu. Par exemple, dans un parc éolien, l'analyse du modèle de détecteur peut révéler si le modèle identifie correctement les défaillances potentielles des engrenages sur la base de modèles de vibrations anormaux. Ou, si le modèle déclenche avec précision des alertes de maintenance lorsque la vitesse du vent dépasse les seuils d'exploitation sûrs. En affinant un modèle basé sur l'analyse, vous pouvez améliorer la maintenance prédictive, réduire les temps d'arrêt et améliorer l'efficacité globale de la production d'énergie.

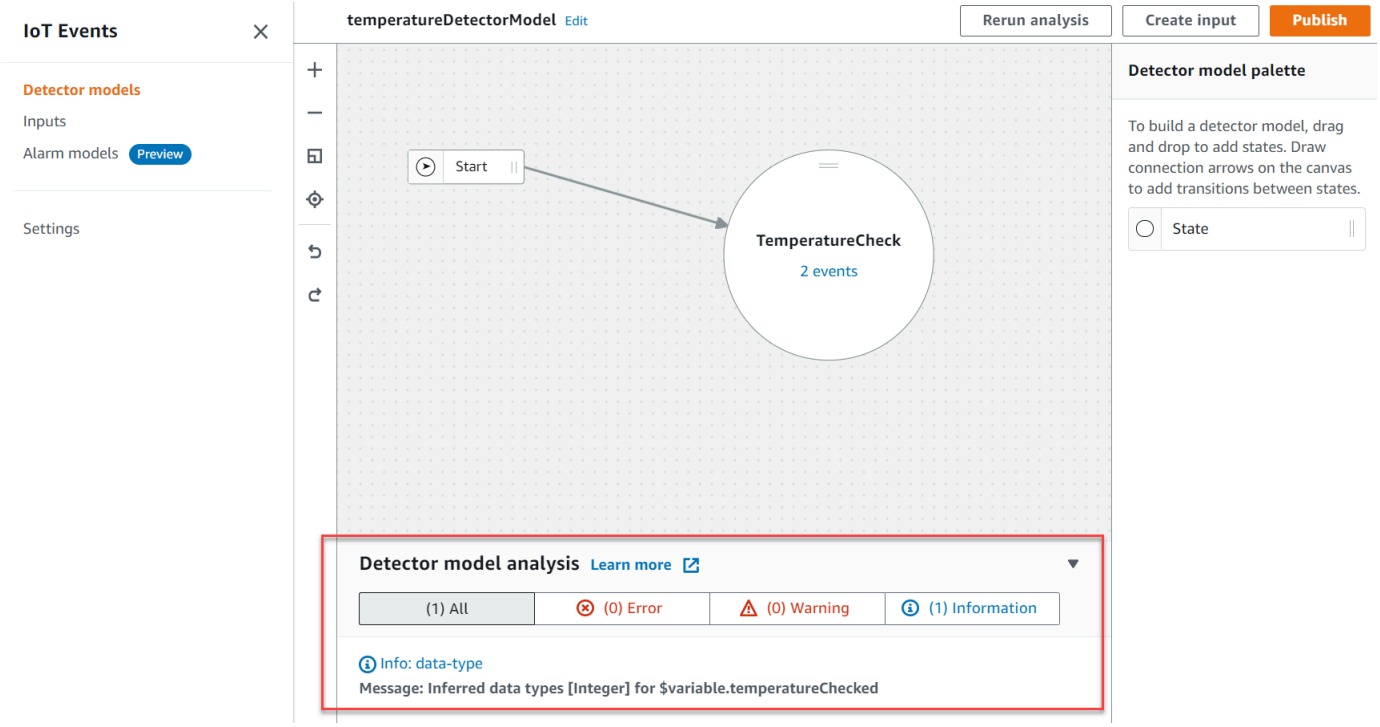
Pour analyser un modèle de détecteur

1. Connectez-vous à la console AWS IoT Events.
2. Dans le volet de navigation, sélectionnez Modèles de détecteurs.

3. Sous Modèles de détecteurs, choisissez le modèle de détecteur cible.
4. Sur la page du modèle de votre détecteur, choisissez Modifier.
5. Dans le coin supérieur droit, sélectionnez Exécuter l'analyse.



Voici un exemple de résultat d'analyse dans la AWS IoT Events console.



Analyser un modèle de détecteur dans AWS IoT Events (AWS CLI)

L'analyse programmatique de vos modèles de AWS IoT Events détecteurs fournit des informations précieuses sur leur structure, leur comportement et leurs performances. Cette approche basée

sur des API permet une analyse automatisée, une intégration à vos flux de travail existants et la possibilité d'effectuer des opérations en masse sur plusieurs modèles de détecteurs. En tirant parti de l'StartDetectorModelAnalysisAPI, vous pouvez lancer des examens approfondis de vos modèles, vous aider à identifier les problèmes potentiels, à optimiser les flux logiques et à garantir que le traitement de vos événements IoT correspond aux exigences de votre entreprise.

Les étapes suivantes utilisent le AWS CLI pour analyser un modèle de détecteur.

Pour analyser un modèle de détecteur à l'aide de AWS CLI

1. Exécutez la commande suivante pour démarrer une analyse.

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

> **Note**
>
> Remplacez *file-name* par le nom du fichier contenant la définition du modèle de détecteur.

Example Définition du modèle de détecteur

```
{
    "detectorModelDefinition": {
        "states": [
            {
                "stateName": "TemperatureCheck",
                "onInput": {
                    "events": [
                        {
                            "eventName": "Temperature Received",
                            "condition":
 "isNull($input.TemperatureInput.sensorData.temperature)==false",
                            "actions": [
                                {
                                    "iotTopicPublish": {
                                        "mqttTopic": "IoTEvents/Output"
                                    }
                                }
                            ]
                        }
```

```
                    ],
                    "transitionEvents": []
                },
                "onEnter": {
                    "events": [
                        {
                            "eventName": "Init",
                            "condition": "true",
                            "actions": [
                                {
                                    "setVariable": {
                                        "variableName": "temperatureChecked",
                                        "value": "0"
                                    }
                                }
                            ]
                        }
                    ]
                },
                "onExit": {
                    "events": []
                }
            }
        ],
        "initialStateName": "TemperatureCheck"
    }
}
```

Si vous utilisez le AWS CLI pour analyser un modèle de détecteur existant, choisissez l'une des options suivantes pour récupérer la définition du modèle de détecteur :

- Si vous souhaitez utiliser la AWS IoT Events console, procédez comme suit :
  1. Dans le volet de navigation, sélectionnez Modèles de détecteurs.
  2. Sous Modèles de détecteurs, choisissez le modèle de détecteur cible.
  3. Choisissez Exporter le modèle de détecteur dans Action pour télécharger le modèle de détecteur. Le modèle du détecteur est enregistré au format JSON.
  4. Ouvrez le fichier JSON du modèle de détecteur.
  5. Vous n'avez besoin que de l'`detectorModelDefinition`objet. Supprimez les éléments suivants :
     - Le premier crochet ({) en haut de la page

- La `detectorModel` ligne
- Objet `detectorModelConfiguration`.
- Le dernier crochet (}) en bas de page

6. Enregistrez le fichier.

- Si vous souhaitez utiliser le AWS CLI, procédez comme suit :

1. Exécutez la commande suivante dans un terminal.

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

2. Remplacez *detector-model-name* par le nom de votre modèle de détecteur.
3. Copiez l'`detectorModelDefinition`objet dans un éditeur de texte.
4. Ajoutez des crochets ({}) à l'extérieur du`detectorModelDefinition`.
5. Enregistrez le fichier au format JSON.

Example Exemple de réponse

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

2. Copiez l'ID d'analyse à partir de la sortie.
3. Exécutez la commande suivante pour récupérer le statut de l'analyse.

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

> **Note**
>
> *analysis-id*Remplacez-le par l'ID d'analyse que vous avez copié.

Example Exemple de réponse

```
{
    "status": "COMPLETE"
}
```

Le statut peut avoir l'une des valeurs suivantes :

- `RUNNING`— AWS IoT Events analyse votre modèle de détecteur. Ce processus peut prendre jusqu'à une minute.
- `COMPLETE`— vous AWS IoT Events avez terminé l'analyse de votre modèle de détecteur.
- `FAILED`— AWS IoT Events impossible d'analyser le modèle de votre détecteur. Réessayez ultérieurement.

4. Exécutez la commande suivante pour récupérer un ou plusieurs résultats d'analyse du modèle de détecteur.

> **Note**
>
> *`analysis-id`*Remplacez-le par l'ID d'analyse que vous avez copié.

```
aws iotevents get-detector-model-analysis-results --analysis-id "analysis-id"
```

Example Exemple de réponse

```
{
    "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**
>
> Après avoir AWS IoT Events commencé à analyser votre modèle de détecteur, vous avez jusqu'à 24 heures pour récupérer les résultats de l'analyse.

AWS IoT Events commandes

Ce chapitre fournit un guide complet de toutes les opérations d'API disponibles dans AWS IoT Events. Il fournit des explications détaillées, notamment des exemples de demandes, de réponses et d'erreurs potentielles pour chaque opération sur les protocoles de services Web pris en charge. La compréhension de ces opérations d'API vous permet de les AWS IoT Events intégrer efficacement à vos applications IoT et d'automatiser vos flux de travail de détection et de réponse aux événements.

AWS IoT Events actions

Vous pouvez utiliser les commandes de AWS IoT Events l'API pour créer, lire, mettre à jour et supprimer des entrées et des modèles de détecteurs, ainsi que pour répertorier leurs versions. Pour plus d'informations, consultez les actions et les types de données pris en charge AWS IoT Events dans le Guide de référence des AWS IoT Events API.

Les AWS IoT Events sections de la référence des AWS CLI commandes incluent les AWS CLI commandes que vous pouvez utiliser pour administrer et manipuler AWS IoT Events.

AWS IoT Events données

Vous pouvez utiliser les commandes de l'API AWS IoT Events Data pour envoyer des entrées aux détecteurs, répertorier les détecteurs et afficher ou mettre à jour l'état d'un détecteur. Pour plus d'informations, consultez les actions et les types de données pris en charge par AWS IoT Events Data dans la référence AWS IoT Events d'API.

Les sections de AWS IoT Events données de la référence des AWS CLI commandes incluent les AWS CLI commandes que vous pouvez utiliser pour traiter AWS IoT Events les données.

Historique du document pour AWS IoT Events

Le tableau suivant décrit les modifications importantes apportées au guide du AWS IoT Events développeur après le 17 septembre 2020. Pour plus d'informations sur les mises à jour de cette documentation, vous pouvez vous abonner à un flux RSS.

Modification	Description	Date
Avis de fin de support	Avis de fin de support : le 20 mai 2026, AWS le support de. AWS IoT Events Après le 20 mai 2026, vous ne pourrez plus accéder à la AWS IoT Events console ni aux AWS IoT Events ressources.	20 mai 2025
Lancement régional	AWS IoT Events est désormais disponible dans la région Asie-Pacifique (Mumbai).	30 septembre 2021
Lancement régional	AWS IoT Events est désormais disponible dans la région AWS GovCloud (ouest des États-Unis).	22 septembre 2021
Résoudre les problèmes liés à un modèle de détecteur en exécutant des analyses	AWS IoT Events peut désormais analyser votre modèle de détecteur et générer des résultats d'analyse que vous pouvez utiliser pour dépanner votre modèle de détecteur.	23 février 2021
Lancement régional	Lancé AWS IoT Events en Chine (Pékin).	30 septembre 2020

Utilisation de l'expression	Ajout d'exemples pour vous montrer comment écrire des expressions.	22 septembre 2020
Surveillance par alarmes	Les alarmes vous aident à surveiller les modifications apportées à vos données. Vous pouvez créer des alarmes qui envoient des notifications lorsqu'un seuil est dépassé.	1er juin 2020

Mises à jour antérieures

Le tableau suivant décrit les modifications importantes apportées au Guide du AWS IoT Events développeur avant le 18 septembre 2020.

Modification	Description	Date
Ajout de la validation de type à la référence Expressions	Des informations de validation de type ont été ajoutées à la référence Expressions.	3 août 2020
Ajout d'un avertissement de région pour les autres services	Ajout d'un avertissement concernant la sélection de la même région pour AWS IoT Events les autres AWS services.	7 mai 2020
Ajouts, mises à jour	• Fonction de personnalisation de la charge utile • Nouvelles actions liées aux événements : Amazon DynamoDB et AWS IoT SiteWise	27 avril 2020

Modification	Description	Date
Fonctions intégrées ajoutées pour les expressions condition nelles du modèle de détecteur	Fonctions intégrées ajoutées pour les expressions condition nelles du modèle de détecteur.	10 septembre 2019
Exemples de modèles de détecteurs ajoutés	Ajout d'exemples pour le modèle de détecteur.	5 août 2019
Ajout de nouvelles actions événementielles	Ajout de nouvelles actions événementielles pour : • Lambda • Amazon SQS • Kinesis Data Firehose • AWS IoT Events entrée	19 juillet 2019
Ajouts, corrections	• Description de la `timeout()` fonction mise à jour. • Ajout d'une meilleure pratique concernant l'inactiv ité des comptes.	11 juin 2019
Politique d'autorisation et options de débogage de console mises à jour	• Mise à jour de la politique d'autorisation de la console. • Image de la page des options de débogage de la console mise à jour.	5 juin 2019
Mises à jour	AWS IoT Events service ouvert à la disponibilité générale.	30 mai 2019

Modification	Description	Date
Ajouts, mises à jour	• Informations de sécurité mises à jour. • Exemple de modèle de détecteur annoté ajouté.	22 mai 2019
Exemples ajoutés et autorisations requises	Ajout d'exemples de charge utile Amazon SNS ; ajouts aux autorisations requises pour. `CreateDetectorModel`	17 mai 2019
Informations de sécurité supplémentaires ajoutées	Informations ajoutées à la section sécurité.	9 mai 2019
Version préliminaire limitée	Version préliminaire limitée de la documentation.	28 mars 2019